▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 25

PASO A PASO



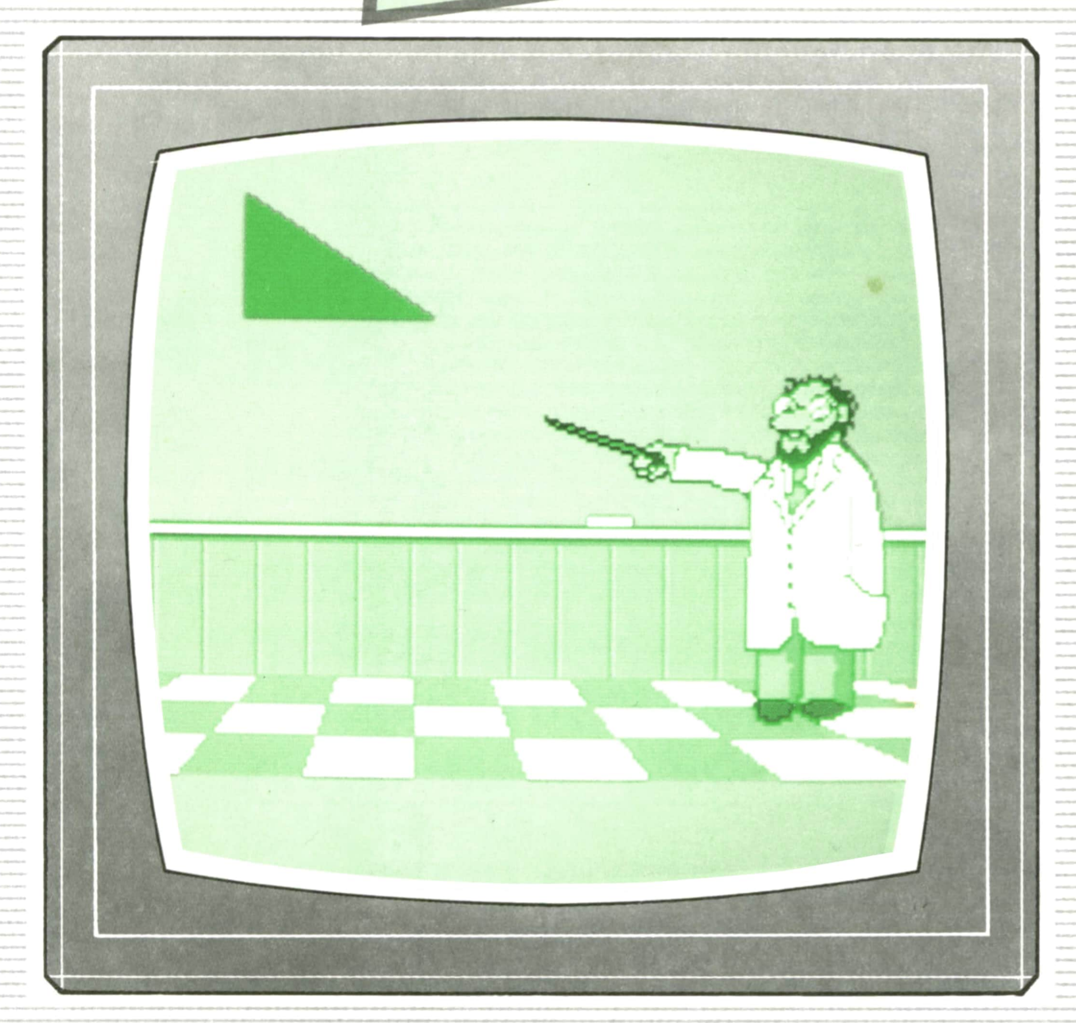
PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 25

Paso a paso



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

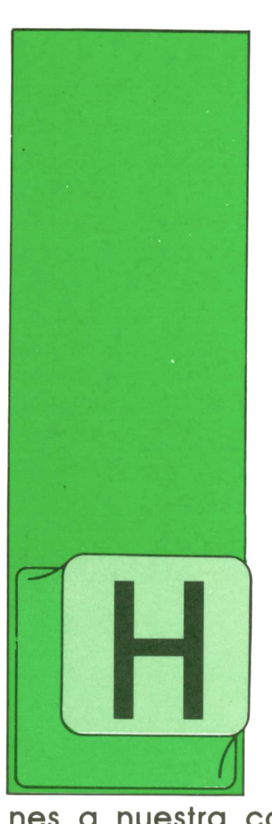
TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas, Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de Aula de Informática Aplicada (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 5.677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.
Octubre, 1987.
P.V.P. Canarias: 335,–.

INDICE



BASIC

FUNCIONES (III)

Funciones definidas por el usuario

HASTA ahora hemos visto las funciones disponibles en los ordenadores que vienen ya definidas en memoria. Sin embargo, nosotros podemos definir nuevas funciones a nuestra conveniencia, de modo que el BASIC que ya conocemos cobra una gran potencia.

Para abordar el estudio de estas funciones vamos a plantear dos etapas:

— Definición de funciones.
— Utilización de funciones definidas.

Definición de funciones

Definir una función significa especificar en el programa dónde deseamos utilizar dicha función, cuál es su objetivo o, más concretamente, cómo se efectúa la función en cuestión. Por tanto, debemos «enseñar» al ordenador a realizarla. Existe una instrucción BASIC cuyo objetivo es precisamente éste, la definición de funciones. Su formato general es el siguiente:

DEF FN <nombre> (<argumento 1>, <argumento 2>,...) = <expresión de definición>

Todos los ordenadores disponen de la instrucción DEF FN, aunque, como veremos después, la versatilidad de la misma varía mucho de unos a otros.

Analicemos detenidamente las distintas partes que componen esta instrucción:

La palabra DEF indica que a continuación se va a definir una función, por tanto, tras DEF se sitúa el nombre de la función, que se compone, a su vez, de dos partes:

— La palabra FN, abreviatura de función.
— El nombre específico, que diferencia unas funciones de otras.

Por tanto, el nombre completo de la función será siempre del tipo:

FN<nombre>

Las reglas para la formación del nombre específico varían de unas versiones de BASIC a otras. Por lo general, siguen las mismas que los nombres de variables, aunque el SPECTRUM sólo admite una letra.

Por otra parte, algunos ordenadores permiten definir funciones numéricas y alfanuméricas, identificando estas últimas del mismo modo que las variables alfanuméricas, es decir, añadiendo el signo $ al final del nombre.

Asimismo, en algunos casos se pueden diferenciar funciones numéricas de los diversos tipos ya conocidos: enteras, reales, simple precisión, etc., añadiendo al final del nombre el signo correspondiente.

De modo que, por ejemplo, FNE sería un nombre de función numérica definida, mientras que FNJ$ sería qué nombre válido de función alfanumérica, en los ordenadores que permitan definir este tipo de funciones.

A continuación del nombre de la función definida se escriben entre paréntesis el o los argumentos que vaya a utilizar.

El número y tipo de argumentos que se pueden especificar varía mucho de unos ordenadores a otros. Algunos sólo admiten un único argumento numérico, mien-

tras que otros podemos utilizar varios argumentos de tipo numérico o alfanumérico o ambos simultáneamente.

Por último, tras el signo igual (=) debemos escribir una expresión en la que detallemos las operaciones y funciones que deberá realizar el ordenador con los argumentos especificados para obtener el resultado deseado.

Veamos algunos ejemplos:

DEF FNR(N) = N^(1/3)

Hemos definido una función numérica de nombre FNR, que utiliza un único argumento numérico, representado por N. El objetivo de la función es realizar la raíz cúbica de un número dado, por tanto, en la expresión de definición se eleva el argumento a 1/3, que es el método para expresar una raíz cúbica en forma de potencia.

DEF FNC(P$) = INT ((40−LEN(P$))/2)

Esta nueva función es de naturaleza numérica y su nombre es FNC. Acepta un solo argumento de tipo alfanumérico representado por P$. Su objetivo es calcular el número de espacios que debemos imprimir delante de la cadena especificada en el argumento, de modo que dicha cadena aparezca centrada en la pantalla. La función se ha definido para una pantalla de 40 columnas, por tanto, si nuestra pantalla es de distinto tamaño habrá que sustituir el 40 por el número de columnas correspondientes.

DEF FNL$ (A$,B$,P) = LEFT$(A$,P−1) + +B$+RIGHT$(A$,LEN(A$)−(P+1)

Por último, esta función, mucho más complicada y no definible en todos los ordenadores, es de tipo alfanumérico y su nombre es FNL$. Utiliza tres argumentos: dos alfanuméricos y uno numérico. El objetivo es insertar la cadena representada por el argumento B$ dentro de la cadena representada por A$ en la posición especificada en P.

Para ello fragmentamos la cadena A$ por la izquierda hasta la posición P−1. A continuación le sumamos la cadena B$ y, finalmente, añadimos el resto de la cadena A$ fragmentado por la derecha.

Hasta aquí hemos visto la definición de funciones, veamos ahora cómo usarlas.

Utilización de funciones definidas

Para utilizar una función definida no tenemos más que escribir el nombre que le hayamos asignado con los argumentos concretos sobre los que deseemos aplicarla.

Veamos un ejemplo. El programa 1 utiliza la función que definimos anteriormente para calcular la raíz cúbica de cualquier número introducido por el teclado.

```
10 REM ********************
20 REM *  RAICES CUBICAS  *
30 REM ********************
40 CLS
50 DEF FNR(N)=N^(1/3)
60 INPUT "DIME UN NUMERO ";A
70 CLS
80 IF A<0 THEN PRINT "POSITIVO, POR FAVOR":GOTO 60
90 LET C=FNR(A)
100 PRINT "NUMERO :",A
110 PRINT :PRINT
120 PRINT "RAIZ CUBICA :",C
```

Es importante observar que el argumento que indicamos al utilizar la función no tiene por qué coincidir con el usado en la definición, ya que este últi-

mo es un argumento genérico, que tiene como misión indicar la naturaleza numérica o alfanumérica.

Veamos otros ejemplos. El programa 2 utiliza el segundo ejemplo de función definida para centrar cualquier texto en la pantalla.

```
10 REM ************************
20 REM *  CENTRADO DE TEXTOS   *
30 REM ************************
40 CLS
50 DEF FNC(P$)=INT((40-LEN(P$))/2)
60 INPUT "TECLEA EL TEXTO A CENTRAR ";T$
70 CLS
80 IF LEN(T$)>38 THEN PRINT "ESE TEXTO ES
   DEMASIADO LARGO PARA CENTRARLO":GOTO 60
90 LET N=FNC(T$)
100 PRINT TAB(N+1);T$
```

Recordemos que este programa sólo es válido para aquellos ordenadores que admitan funciones definidas con argumentos alfanuméricos.

El programa 3 aplica la función que definimos anteriormente para insertar una cadena dentro de otra.

```
10 REM **************************
20 REM *  INSERCION DE CADENAS   *
30 REM **************************
40 CLS
50 DEF FNL$(A$,B$,P)=LEFT$(A$,P-1)+B$+RIGHT$(A$,LEN(A$)-P+1)
60 LET F$="YO  TODOS LOS DIAS"
70 PRINT F$
80 PRINT :PRINT :PRINT
90 INPUT "ESCRIBE UN VERBO QUE COMPLETE ESTA FRASE ";V$
100 LET N$=FNL$(F$,V$,4)
110 CLS
120 PRINT N$
```

En este caso concreto, la función se encarga de insertar el verbo que deseemos dentro de la frase asignada a la variable F$.

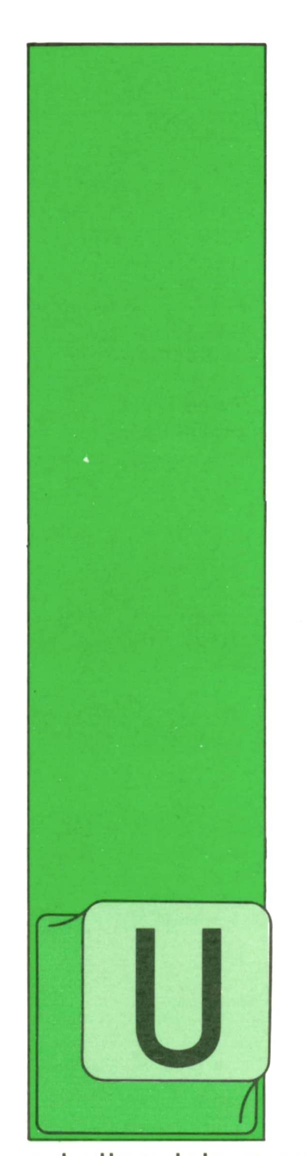
Este programa sólo será válido para los ordenadores en los que se puedan definir funciones alfanuméricas con argumentos alfanuméricos.

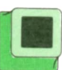
Por último, hay que matizar que la instrucción DEF FN sólo sirve para indicar lo que hará la función al ser utilizada, pero el ordenador no realizará cálculo alguno al leer DEF FN.

Finalmente, en la tabla de la figura 1 se resumen las características de las funciones definidas para distintos ordenadores.

	Nombre de la función	Número argumentos	Nombre de argumento(s)	Naturaleza de la función	Naturaleza de argumento(s)
AMSTRAD	Como variables	Varios	Como variables	Numérica Alfanumérica	Numérico Alfanumérico
COMMODORE	Como variables	Uno	Como variables	Numérica	Numérico
IBM	Como variables	Varios	Como variables	Numérica Alfanumérica	Numérico Alfanumérico
MSX	Como variables	Varios	Como variables	Numérica Alfanumérica	Numérico Alfanumérico
SPECTRUM	Una letra	Varios	Una letra	Numérica Alfanumérica	Numérico Alfanumérico

MAQUINA 8088





Interrupciones

UNA interrupción es el proceso que realiza la CPU cuando recibe una petición urgente de realizar una tarea específica.

Cuando la CPU recibe una petición de este tipo, interrumpe provisionalmente la tarea que está realizando y pasa a ejecutar la tarea requerida. Una vez terminada, reanuda la tarea interrumpida en la misma situación en que la abandonó.

Los mecanismos de interrupciones surgieron para aumentar la eficacia de la CPU en el manejo de los dispositivos periféricos (teclado, discos, impresoras, etcétera) y están totalmente implementados por hardware.



Necesidad de las interrupciones

Supongamos por un momento que, utilizando un sistema que no dispone de un mecanismo de interrupciones, queremos escribir un programa que use el teclado, la pantalla y un fichero en diskette.

El programa debe funcionar de forma que cada vez que se pulse una tecla, debe guardarse el carácter en un área de memoria y representarlo en la posición adecuada de la pantalla.

Una rutina que realice esta tarea podría hacerse con un bucle compuesto por las siguientes operaciones:

1. Esperar una acción sobre el teclado.
2. Leer el carácter pulsado.
3. Salir del bucle si se ha leído el carácter de terminación.
4. Escribir dicho carácter sobre la pantalla.
5. Guardar el carácter en memoria.
6. Volver al punto 1.

La rutina se podría escribir de esta forma. Pero, ¿qué pasaría cuando se deja de teclear?

Evidentemente, la CPU quedaría esperando en la instrucción 1 del bucle hasta que se volviera a actuar sobre el teclado. Durante todo ese tiempo, la CPU estaría parada.

Veamos ahora la situación que se produce cuando el programa tiene que escribir en el diskette. La rutina básica de escritura debe hacer lo siguiente.

1. Mandar posicionar la cabeza de escritura sobre la pista adecuada.

2. Esperar a que la cabeza esté posicionada.

3. Esperar a que el origen de la pista pase por la cabeza de escritura.

4. Transferir los datos desde la memoria al diskette.

Las esperas de los puntos 2 y 3 pueden conseguirse con bucles en los que se esté continuamente comprobando cuándo se alcanza el objetivo deseado, es decir, el posicionamiento de la cabeza o el paso del origen de la pista.

Si usamos rutinas como las descritas, ¿qué pasaría si un programa quisiera simultanear tareas de lectura de teclado con escritura en diskette?

En ese caso, mientras se está realizando el bucle de espera del teclado no se puede realizar el bucle de espera del diskette y viceversa, por lo cual, el programa debería realizar primero una tarea y después la otra. A menos que se escribiera una rutina específica que en el mismo bucle atendiera al teclado y al diskette.

La misma situación se presentaría si en otro programa se quisiera leer del diskette para escribir en impresora o en cualquier otra combinación de periféricos.

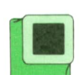
Llegamos a la conclusión de que en un sistema sin mecanismo de interrupciones, una de dos:

— O se pierde eficacia en el manejo de periféricos.

— O hay que programar las rutinas básicas para cada situación concreta.

Esta última alternativa puede servir para programas aislados, pero no para escribir las rutinas básicas de manejo de periféricos propias de un sistema operativo, y, por supuesto, no permite ni el más elemental grado de multiprogramación.

¿Cómo puede resolver este problema un sistema de interrupciones? Una CPU con interrupciones puede programarse de forma que los tiempos de espera de unas tareas puedan emplearse para realizar otras, de forma que al producirse el suceso que hace terminar la espera (la pulsación de un nuevo carácter o la llegada de la cabeza de escritura a la pista deseada), se ejecute automáticamente el código correspondiente.



Las interrupciones del 8088

La familia de microprocesadores 8088 tiene un sistema de interrupciones simple y versátil que le permite manejar hasta 256 interrupciones. Las peticiones de interrupción pueden ser:

Externas. Solicitadas por algunos de los dispositivos externos a la CPU.

Internas. Generadas por la propia actividad de la CPU.

Hay dos tipos diferentes de **interrupciones externas,** cuyas peticiones llegan a la CPU por dos hilos independientes denominados:

INTR. Abreviatura de «Interrupt Request», que significa petición de interrupción.

NMI. Abreviatura de «non maskable interrupt» que significa: Interrupción que no puede esperar.

Los sucesos que pueden ocasionar interrupciones **NMI,** son típicamente sucesos «catastróficos», como alteración de la tensión de alimentación, error de memoria o error de paridad en el bus.

Por el contrario, por el hilo **INTR** llegan a la CPU las interrupciones que, derivadas de la actividad normal, se originan en los periféricos.

Para saber en cada caso concreto cuál ha sido el periférico que ha provocado la interrupción, el 8088 utiliza los servicios de un circuito auxiliar denominado PIC (Intel 8259A, Programmable Interrupt Controller).

El PIC actúa a modo de «secretario de interrupciones» del 8088 y tiene encomendadas las siguientes misiones:

— Aceptar del 8088 y memorizar las prioridades que debe asignar a cada una de las interrupciones que controla.

— Recibir físicamente las peticiones de interrupción de los distintos periféricos a través de hilos diferentes.

— Avisar al 8088, a través del hilo INTR, cuando se produce alguna petición de interrupción.

— Comunicar al 8088, cuando éste se lo pide, el código (un byte) que permite distinguir las diferentes interrupciones.

— En caso de acumularse varias peticiones, ir avisando sucesivamente al 8088 de las interrupciones pendientes, empezando por las más prioritarias.

Además de todo lo dicho, el 8088 dispone de un flag denominado IF (interrupt enable flag) con el cual se puede anular el mecanismo de interrupciones externas durante intervalos en los que se realizan tareas que no se deben interrumpir.

Las **interrupciones internas** utilizan el mismo mecanismo diseñado para atender a los periféricos para interrumpir el código que se está ejecutando en un momento dado, pasar a ejecutar otro y volver al punto en que se interrumpió.

Se provocan interrupciones internas en los siguientes casos:

— Por instrucciones INT (interrupt).

— Por la instrucción INTO (interrupt on overflow).

— Por las instrucciones DIV (divide) o IDIV (integer divide) si el cociente resultante no cabe en el registro al que va destinado.

— Al terminar de ejecutarse cada instrucción, si está activo el flag TF (trap flag). Esta situación la provocan los programas de puesta a punto, que como el DEBUG (citado con anterioridad), son capaces de controlar la ejecución de otro programa en la modalidad «paso a paso».



El vector de interrupciones

Hemos hablado hasta ahora de los distintos tipos de interrupciones, pero, ¿cómo conoce la CPU la dirección del código que debe atender a cada una de las 256 posibles interrupciones?

Un sencillo convenio resuelve este problema. La CPU se reserva en el origen de la memoria una tabla de 256 entradas, en las cuales supone que se han colocado los apuntadores a los códigos que atienden a las 256 interrupciones posibles.

Cada apuntador, que tiene una longitud de 4 bytes, se compone del segmento de código y un desplazamiento y puede, por tanto, apuntar a cualquier zona de memoria de la máquina. Por lo cual la tabla tiene una longitud de 1 Kbyte (4 × 256 = 1.024 bytes).

En el funcionamiento normal, el sistema operativo se encarga de situar en memoria los códigos que atienden a las interrupciones y de definir en la posición adecuada de esta tabla el apuntador que direcciona dicho código.

En la siguiente figura se da una lista de las interrupciones utilizadas, las reservadas y las que quedan libres. La lista está compuesta por las siguientes cinco columnas:

— La primera columna, titulada «Num», indica el número de la interrupción expresado en decimal.

— La segunda columna, titulada «Direc», contiene la dirección del apuntador asociado a esa interrupción.

— La tercera columna, titulada «Causa», indica cuál puede ser la causa de que se desencadene la interrupción.

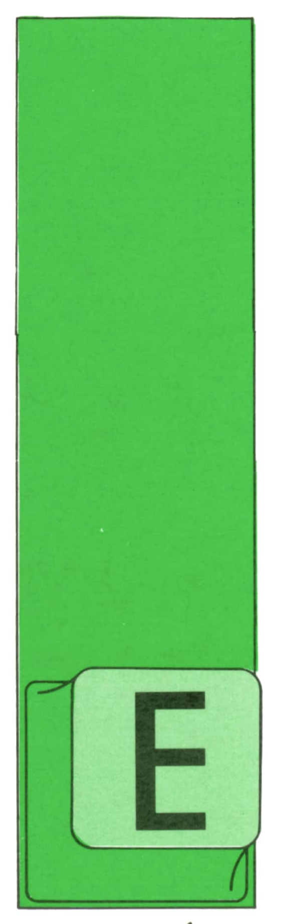
— La cuarta columna, encabezada por la palabra «Nivel», indica qué componente de la máquina es el que ha establecido esa interrupción.

— En la última columna se describe brevemente de qué trata la interrupción.

Núm.	Direc.	Causa	Nivel	Descripción
0	0000	DIV, IDIV	8088	Error en la división.
1	0004	Flag TF	8088	Interrupción paso a paso.
2	0008	Hardware	8088	NMI. (Errores no recuperables).
3	000C	INT 3	8088	Punto de control en puesta a punto.
4	0010	INTO	8088	Error en operación aritmética (Overflow).
5	0014	INT 5	BIOS	Copia de la pantalla a impresora.
6	0018		BIOS	Reservada.
7	001C		BIOS	Reservada.
8	0020	Reloj	BIOS	Reloj interno 18,2 pulsos por segundo.
9	0024	Teclado	BIOS	Interrupción del hardware del teclado.
10	0028		BIOS	Reservada.
11	002C		BIOS	Reservada.
12	0030	RS232	BIOS	Interrupción del hardware de comunicaciones.
13	0034		BIOS	Reservada.
14	0038	Diskette	BIOS	Interrupción del hardware de disco/diskette.
15	003C	Impresora	BIOS	Interrupción del hardware de la impresora.
16	0040	INT 10H	BIOS	Funciones de utilización de pantallas.
17	0044	INT 11H	BIOS	Comprobación del equipo instalado.
18	0048	INT 12H	BIOS	Determinación de la memoria instalada.
19	004C	INT 13H	BIOS	Funciones de utilización de discos/diskettes.
20	0050	INT 14H	BIOS	Funciones de utilización de comunicaciones.
21	0054	INT 15H	BIOS	Funciones de utilización de cassettes.
22	0058	INT 16H	BIOS	Funciones de utilización del teclado.
23	005C	INT 17H	BIOS	Funciones de utilización de la impresora.
24	0060	INT 18H	BIOS	Funciones de utilización de cassette en BASIC.
25	0064	INT 19H	BIOS	Arranque del programa inicial. (Boostrap).
26	0068	INT 1AH	BIOS	Fecha y hora.
27	006C	INT 1B	BIOS	Acción de usuario en caso de Ctrl-Break.
28	0070	INT 1C	BIOS	Acción de usuario en los pulsos de reloj.
29	0074	INT 1D	BIOS	Inicialización de parámetros de pantalla.
30	0078	INT 1E	BIOS	Inicialización de parámetros de diskette.
31	007C	INT 1F	BIOS	Tabla de caracteres en modo gráfico.
32	0080	INT 20H	DOS	Terminación de programa.
33	0084	INT 21H	DOS	Funciones DOS ("calls").
34	0088	INT 22H	DOS	Dirección de terminación de un programa.
35	008C	INT 23H	DOS	Acción a realizar por pulsar Ctrl-Break.
36	0090	INT 24H	DOS	Error crítico.
37	0094	INT 25H	DOS	Lectura absoluta del disco o diskette.
38	0098	INT 26H	DOS	Escritura absoluta en el disco o diskette.
39	009C	INT 27H	DOS	Terminación de un programa residente.
40	00A0	INT 28H	DOS	Reservadas para DOS.
...	...	...	...	...
63	00FC	INT 3FH	DOS	Reservadas para DOS.
64	0100	INT 40H		No usadas.
...	...	...	...	...
127	01FC	INT 7FH		No usadas.
128	0200	INT 80H	BASIC	Reservadas para BASIC.
...	...	...	...	...
240	03C0	INT 0F0H	BASIC	Reservadas para BASIC.
241	03C4	INT 0F1H		No usadas.
...	...	...	...	...
255	03FC	INT 0FFH		No usadas.

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS





Programa: PAC-MAN para IBM

EL PAC-MAN es uno de los juegos más famosos que existen para ordenador. Uno de los más famosos y de los más sencillos de utilizar. Todo el juego consiste en comerse el mayor número de puntos posibles, sin que los fantasmas que habitan el laberinto nos coman a nosotros.

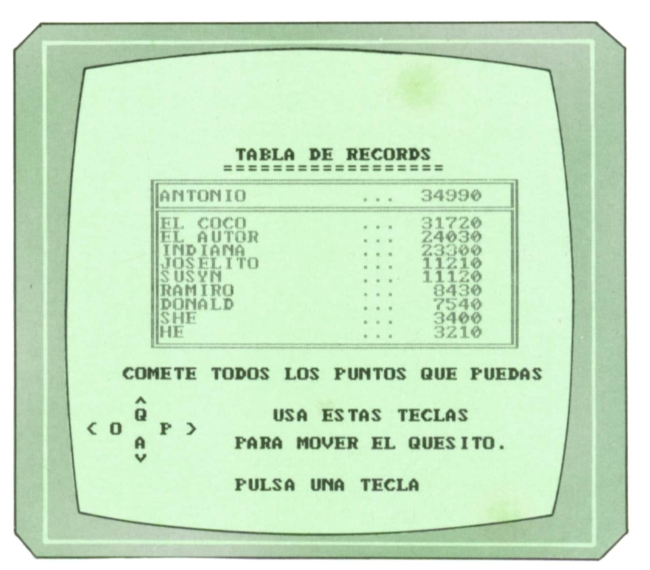
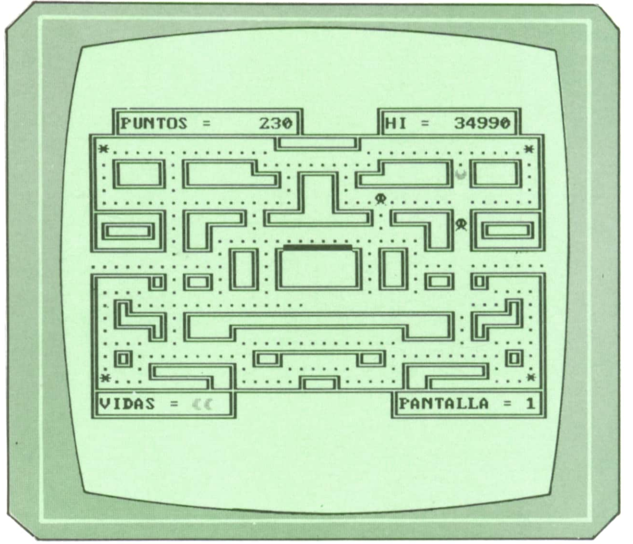



Tabla de récords e ilustraciones.

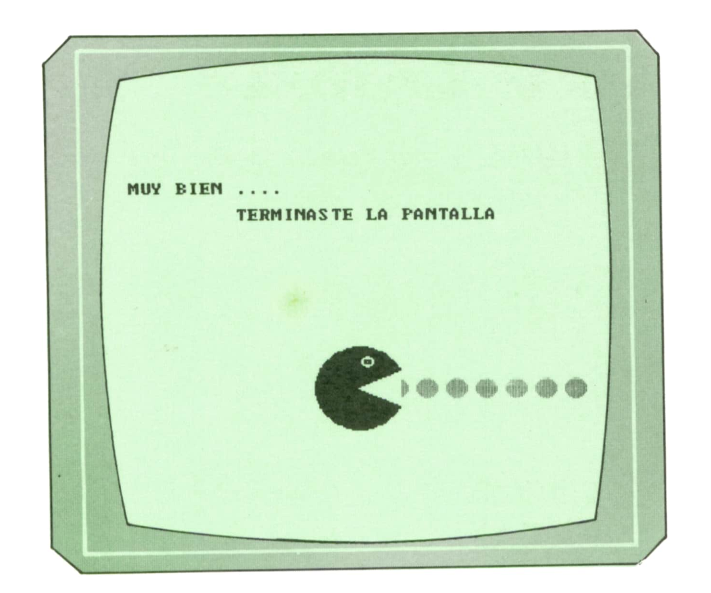



Pantalla de presentación.




La pantalla de juego.

La única diferencia de este programa con el original es que cuando nos comemos los puntos gordos, no podemos comernos los fantasmas. A cambio, cada vez que nos comamos un asterisco, podremos movernos de prisa.




```
1000 REM ************************
1010 REM *                      *
1020 REM *    P A C  -  M A N   *
1030 REM *                      *
1040 REM ************************
1050 REM
1060 REM ************************
1070 REM *                      *
1080 REM *       A U T O R      *
1090 REM *      -----------     *
1100 REM *                      *
1110 REM * Fco. Morales Guerrero *
1120 REM *                      *
1130 REM ************************
1140 REM
1150 REM *************************
1160 REM *                       *
1170 REM * (c) Ed. Siglo Cultural *
1180 REM * (c) 1987.             *
1190 REM *                       *
1200 REM *************************
1210 REM
1220 REM ******************************
1230 REM * INICIALIZACION DEL PROGRAMA *
1240 REM ******************************
1250 REM
1260 DIM A1%(10),A2%(10),A3%(10),A4%(10),B%(10),C%(10),D%(10),N(9),N$(10),P(10),
S(25,41),X%(409),Y%(435),M$(2,3),I(2,8)
1270 M$(1,0)=".":M$(1,1)="*":M$(1,3)=" ":M$(2,0)=".":M$(2,1)="*":M$(2,3)=" "
1280 PA=1
1290 SCREEN 1:WIDTH 40:KEY OFF
1300 COLOR ,,,,3
1310 CLS
1320 PRINT TAB(15);"COME - COCOS"
1330 PRINT TAB(14);"=============="
1340 COLOR ,,,,1
1350 LOCATE 10,14:PRINT "CARGANDO DATAS"
1360 COLOR ,,,,2
1370 LOCATE 12,12:PRINT "ESPERA UN MOMENTO."
1380 REM
1390 REM *******************************************
1400 REM * LECTURA DE LAS FIGURAS DE LOS PERSONAJES *
1410 REM *******************************************
1420 REM
```

```
1430 GET (0,0)-(7,7),A1%
1440 GET (0,0)-(7,7),A2%
1450 GET (0,0)-(7,7),A3%
1460 GET (0,0)-(7,7),A4%
1470 GET (0,0)-(7,7),B%
1480 GET (0,0)-(7,7),C%
1490 GET (0,0)-(7,7),D%
1500 GET (105,75)-(167,125),X%
1510 RESTORE 5920
1520 FOR I=0 TO 10
1530     READ A1%(I)
1540     READ B%(I)
1550 NEXT I
1560 FOR I=0 TO 10
1570     READ C%(I)
1580     READ D%(I)
1590 NEXT I
1600 FOR I=0 TO 409
1610     READ X%(I)
1620 NEXT I
1630 FOR I=0 TO 435
1640     READ Y%(I)
1650 NEXT I
1660 FOR I=0 TO 10
1670     READ A2%(I)
1680     READ A3%(I)
1690     READ A4%(I)
1700 NEXT I
1710 REM
1720 REM *****************************
1730 REM * LECTURA DE LOS CARACTERES *
1740 REM *****************************
1750 REM
1760 FOR I=0 TO 9
1770     READ N(I)
1780 NEXT I
1790 REM
1800 REM **************************
1810 REM * LECTURA DE LOS RECORDS *
1820 REM **************************
1830 REM
1840 FOR I=1 TO 10
1850     READ N$(I),P(I)
1860 NEXT I
1870 REM
1880 REM ******************************
1890 REM * LECTURA DE LAS DIRECCIONES *
1900 REM ******************************
1910 REM
1920 FOR I=1 TO 2
1930     FOR J=1 TO 8
1940         READ I(I,J)
1950     NEXT J
1960 NEXT I
1970 REM
1980 REM **********************
1990 REM * PROGRAMA PRINCIPAL *
2000 REM **********************
2010 REM
2020 CLS
2030 GOSUB 5380:REM PRESENTACION
2040 GOSUB 4520:REM TABLA DE RECORDS
2050 GOSUB 5130:REM INSTRUCCIONES
2060 GOSUB 5280:REM PULSA UNA TECLA
2070 GOSUB 4180:REM DIBUJA LA PANTALLA
2080 COLOR ,,,,2
2090 GOSUB 2150 :REM LEE EL TECLADO
2100 GOSUB 2300 :REM MUEVE EL COME-COME
2110 GOSUB 2660 :REM ENEMIGO NUMERO 1
```

```
2120 GOSUB 2890 :REM ENEMIGO NUMERO 2
2130 IF PT<=0 THEN GOSUB 3780:PA=PA+1:GOTO 2070
2140 GOTO 2090
2150 REM
2160 REM ***********************
2170 REM * CONTROL DEL TECLADO *
2180 REM ***********************
2190 REM
2200 I$=INKEY$:IF I$="" THEN I$=Z$
2210 Z$=I$
2220 II=INSTR("OoPpQqAa",I$)
2230 IF II=0 THEN RETURN
2240 X1=X0:Y1=Y0
2250 X0=X0+I(1,II)+I(1,II)*(S(Y0,X0+I(1,II))=0)
2260 Y0=Y0+I(2,II)+I(2,II)*(S(Y0+I(2,II),X0)=0)
2270 IF X0=41 THEN X0=1
2280 IF X0=0 THEN LET X0=40
2290 RETURN
2300 REM
2310 REM ****************************
2320 REM * MOVIMIENTO DEL COME-COME *
2330 REM ****************************
2340 REM
2350 IF BB>0 THEN BB=BB+1:IF BB=60-(10*PA) THEN BB=0
2360 IF X1=X0 AND Y1=Y0 THEN Z$="Z":RETURN
2370 IF S(Y0,X0)=-1 THEN PT=PT-1:S(Y0,X0)=2:PU=PU+10:LOCATE 2,13:PRINT USING "##
####";PU
2380 IF S(Y0,X0)=1 THEN PT=PT-1:S(Y0,X0)=2:PU=PU+50:LOCATE 2,13:PRINT USING "###
###";PU:LET BB=1
2390 IF PU>=HI THEN LOCATE 2,32:PRINT USING "######";PU:HI=PU
2400 LOCATE Y1,X1:PRINT " "
2410 S(Y1,X1)=2
2420 IF SW=1 THEN PUT(8*(X0-1),8*(Y0-1)),B%,PSET:SW=0:S(Y0,X0)=99:RETURN
2430 ON II GOSUB 2460,2460,2510,2510,2560,2560,2610,2610
2440 SW=1:S(Y0,X0)=99
2450 RETURN
2460 REM
2470 REM *** MOVIMIENTO A LA IZQUIERDA ***
2480 REM
2490 PUT(8*(X0-1),8*(Y0-1)),A2%,PSET
2500 RETURN
2510 REM
2520 REM *** MOVIMIENTO A LA DERECHA ***
2530 REM
2540 PUT(8*(X0-1),8*(Y0-1)),A1%,PSET
2550 RETURN
2560 REM
2570 REM *** MOVIMIENTO A ARRIBA ***
2580 REM
2590 PUT(8*(X0-1),8*(Y0-1)),A4%,PSET
2600 RETURN
2610 REM
2620 REM *** MOVIMIENTO A ABAJO ***
2630 REM
2640 PUT(8*(X0-1),8*(Y0-1)),A3%,PSET
2650 RETURN
2660 REM
2670 REM ****************************
2680 REM * MOVIMIENTO DEL FANTASMA-1 *
2690 REM ****************************
2700 REM
2710 REM *** CALCULO DE LA DIRECCION ***
2720 REM
2730 S0=0:MM=MM+1
2740 IF BB>0 AND SD1=1 THEN SD1=0:RETURN
2750 SD1=1
2760 X3=X2:Y3=Y2
2770 SX=SGN(X1-X2)
2780 SY=SGN(Y1-Y2)
```

```
2790 X2=X2+SX+SX*(S(Y2,X2+SX)=0)
2800 Y2=Y2+SY+SY*(S(Y2+SY,X2)=0)
2810 REM
2820 REM *** MOVIMIENTO ***
2830 REM
2840 LOCATE Y3,X3:PRINT M$(1,M1+1)
2850 M1=S(Y2,X2)
2860 IF M1=99 THEN GOTO 3140
2870 IF SE1=1 THEN SE1=0:PUT(8*(X2-1),8*(Y2-1)),D%,PSET:RETURN
2880 SE1=1:PUT(8*(X2-1),8*(Y2-1)),C%,PSET:RETURN
2890 REM
2900 REM ******************************
2910 REM * MOVIMIENTO DEL FANTASMA-2 *
2920 REM ******************************
2930 REM
2940 REM *** CALCULO DE LA DIRECCION ***
2950 REM
2960 SO=0
2970 IF BB>0 AND SD2=1 THEN SD2=0:RETURN
2980 SD2=1
2990 IF X3=X4 AND Y3=Y4 THEN MM=9
3000 IF MM<15 THEN PUT(8*(X4-1),8*(Y4-1)),C%,PSET:RETURN
3010 X5=X4:Y5=Y4
3020 SX=SGN(X1-X4)
3030 SY=SGN(Y1-Y4)
3040 X4=X4+SX+SX*(S(Y4,X4+SX)=0)
3050 Y4=Y4+SY+SY*(S(Y4+SY,X4)=0)
3060 REM
3070 REM *** MOVIMIENTO ***
3080 REM
3090 LOCATE Y5,X5:PRINT M$(2,M2+1)
3100 M2=S(Y4,X4)
3110 IF M2=99 THEN GOTO 3140
3120 IF SE2=1 THEN SE2=0:PUT(8*(X4-1),8*(Y4-1)),D%,PSET:RETURN
3130 SE2=1:PUT(8*(X4-1),8*(Y4-1)),C%,PSET:RETURN
3140 REM
3150 REM ***********************
3160 REM * PERDIDA DE UNA VIDA *
3170 REM ***********************
3180 REM
3190 LOCATE Y0,X0:PRINT " "
3200 FOR I=1 TO 10
3210    FOR J=1 TO 2
3220       PUT(8*(X0-1),8*(Y0-1)),A1%
3230       FOR K=1 TO 100
3240       NEXT K
3250    NEXT J
3260 NEXT I
3270 PUT(8*(X0-1),8*(Y0-1)),A1%
3280 FOR I=1 TO A1%(1)
3290    A1%(1)=I
3300    PUT(8*(X0-1),8*(Y0-1)),A1%,XOR
3310    FOR J=1 TO 100
3320    NEXT J
3330 NEXT I
3340 LOCATE Y0,X0:PRINT " "
3350 LOCATE Y2,X2:PRINT " "
3360 LOCATE Y4,X4:PRINT " "
3370 LET NV=NV-1
3380 IF NV=0 THEN GOTO 3490
3390 Y0=16:X0=20:Y1=Y0:X1=X0:Z$="Z":BB=0
3400 X2=19:Y2=11:X3=X2:Y3=Y2:X4=21:Y4=11:X5=X4:Y5=Y4:M1=S(Y2,X2):M2=S(Y4,X4)
3410 LOCATE 24,10:PRINT "    ";
3420 FOR I=1 TO NV-1
3430    PUT(64+8*I,184),A1%
3440 NEXT I
3450 PUT(8*(X0-1),8*(Y0-1)),A1%
3460 GOSUB 3660
3470 RETURN
```

```
3480 RETURN
3490 REM
3500 REM *************
3510 REM * GAME OVER *
3520 REM *************
3530 REM
3540 FOR I=1 TO 5
3550    FOR J=1 TO 3
3560       COLOR ,,,,J
3570       LOCATE 13,18:PRINT " GAME "
3580       LOCATE 14,18:PRINT " OVER "
3590       FOR K=1 TO 140:NEXT K
3600    NEXT J
3610    LOCATE 13,18:PRINT "      "
3620    LOCATE 14,18:PRINT "      "
3630    FOR K=1 TO 140:NEXT K
3640 NEXT I
3650 GOTO 2040
3660 REM
3670 REM ***************
3680 REM * ESTAS LISTO *
3690 REM ***************
3700 REM
3710 LOCATE 13,18:PRINT "(ESTAS"
3720 LOCATE 14,18:PRINT "LISTO?"
3730 FOR J=1 TO 1000
3740 NEXT J
3750 LOCATE 13,18:PRINT "      "
3760 LOCATE 14,18:PRINT "      "
3770 RETURN
3780 REM
3790 REM ***************************
3800 REM * TERMINO DE UNA PANTALLA *
3810 REM ***************************
3820 REM
3830 FOR I=1 TO 10
3840    COLOR 12
3850    FOR J=1 TO 100
3860    NEXT J
3870    COLOR 0
3880    FOR J=1 TO 100
3890    NEXT J
3900 NEXT I
3910 CLS
3920 COLOR ,,,,3
3930 PRINT "MUY BIEN ...."
3940 PRINT
3950 PRINT TAB(10);"TERMINASTE LA PANTALLA"
3960 FOR I=4 TO 15
3970    CIRCLE(I*20,125),7,1
3980    PAINT(I*20,125),1
3990 NEXT I
4000 FOR I=1 TO 51
4010    X%(1)=I
4020    PUT(0,100),X%,PSET
4030 NEXT I
4040 PUT(0,100),X%
4050 FOR I=0 TO 251 STEP 6
4060    PUT(I,100),X%,PSET
4070    FOR J=1 TO 40:NEXT J
4080    PUT(I+3,100),Y%,PSET
4090    FOR J=1 TO 40:NEXT J
4100 NEXT I
4110 PRINT
4120 PRINT " HAS CONSEGUIDO ..."
4130 PRINT
4140 PRINT PA*600;"PUNTOS DE BONUS"
4150 PU=PU+PA*600
4160 GOSUB 5280
```

```
4170 RETURN
4180 REM
4190 REM *************************
4200 REM * DIBUJO DE LA PANTALLA *
4210 REM *************************
4220 REM
4230 CLS
4240 RESTORE 6770
4250 FOR I=1 TO 25
4260    IF I=25 THEN LOCATE 25,1
4270    READ A$
4280    FOR J=1 TO LEN(A$)
4290       S(I,J)=0
4300       LET B$=MID$(A$,J,1)
4310       IF B$="." OR B$="*" THEN COLOR ,,,,2:PRINT B$;:S(I,J)=(B$=".")-(B$="*
"):COLOR ,,,,3:GOTO 4370
4320       IF B$>"9" THEN GOTO 4340
4330       PRINT CHR$(N(VAL(B$)));:GOTO 4370
4340       IF B$="A" THEN PRINT CHR$(202);
4350       IF B$="B" THEN PRINT CHR$(203);
4360       IF B$="C" THEN PRINT CHR$(223);
4370    NEXT J
4380 NEXT I
4390 COLOR ,,,,3
4400 LOCATE 2,4:PRINT USING "PUNTOS = ######";PU
4410 LOCATE 2,27:PRINT USING "HI = ######";HI
4420 LOCATE 24,2:PRINT "VIDAS = ";
4430 LOCATE 24,28:PRINT USING "PANTALLA = #";PA;
4440 FOR I=1 TO NV-1
4450    PUT(64+8*I,184),A1%
4460 NEXT I
4470 PUT(152,120),A1%,PSET
4480 Y0=16:X0=20:Y1=Y0:X1=X0:Z$="Z":S(13,41)=2:S(13,0)=2:BB=0:MM=0
4490 PT=287:X2=19:Y2=11:X3=X2:Y3=Y2:X4=21:Y4=11:X5=X4:Y5=Y4:M1=-1:M2=-1
4500 GOSUB 3660
4510 RETURN
4520 REM
4530 REM ********************
4540 REM * TABLA DE RECORDS *
4550 REM ********************
4560 REM
4570 CLS
4580 COLOR ,,,,3
4590 PRINT TAB(13);"TABLA DE RECORDS"
4600 PRINT TAB(12);"=================="
4610 COLOR ,,,,1
4620 PRINT TAB(6);CHR$(201);STRING$(28,205);CHR$(187)
4630 PRINT TAB(6);CHR$(186);SPC(28);CHR$(186)
4640 PRINT TAB(6);CHR$(204);STRING$(28,205);CHR$(185)
4650 FOR I=1 TO 9
4660    PRINT TAB(6);CHR$(186);SPC(28);CHR$(186)
4670 NEXT I
4680 PRINT TAB(6);CHR$(200);STRING$(28,205);CHR$(188)
4690 NP=0
4700 FOR I=1 TO 10
4710    IF PU>P(I) THEN LET NP=1
4720 NEXT I
4730 IF NP=1 THEN LET P(10)=PU:N$(10)=CHR$(255)
4740 FOR I=1 TO 10
4750    FOR J=1 TO I
4760       IF P(I)>P(J) THEN A$=N$(I):A=P(I):N$(I)=N$(J):P(I)=P(J):N$(J)=A$:P(J)
=A
4770    NEXT J
4780 NEXT I
4790 IF NP=0 THEN GOTO 4840
4800 FOR I=1 TO 10
4810    IF N$(I)=CHR$(255) THEN LET N$(I)=STRING$(15,"-"):LET NP=I
4820 NEXT I
4830 COLOR ,,,,2
```

```
4840 LOCATE 4,7:PRINT N$(1);TAB(23);"... ";USING "######";P(1)
4850 FOR I=2 TO 10
4860    LOCATE 4+I,7
4870    PRINT N$(I);TAB(23);"... ";USING "######";P(I)
4880 NEXT I
4890 IF NP=0 THEN GOTO 5080
4900 COLOR ,,,,3
4910 LOCATE 16,15:PRINT "FELICIDADES."
4920 PRINT
4930 PRINT TAB(3);"TU PUNTUACION ESTA ENTRE LAS MEJORES"
4940 PRINT
4950 PRINT TAB(12);"ESCRIBE TU NOMBRE."
4960 N$(NP)="":X=7:LO=0:D$=""
4970 IF NP=1 THEN LET Y=4 ELSE Y=4+NP
4980 LOCATE Y,X
4990 PRINT " ";CHR$(29);
5000 LET A$=INKEY$:IF A$="" THEN GOTO 5000
5010 IF A$=CHR$(8) AND LO>0 THEN X=X-1:LO=LO-1:D$=LEFT$(D$,LO):PRINT CHR$(29);"
-";CHR$(29);CHR$(29);
5020 IF A$=" " THEN GOTO 5050
5030 IF A$=CHR$(13) THEN GOTO 5060
5040 IF A$<"0" OR A$>"z" THEN GOTO 5000
5050 PRINT A$;" ";CHR$(29);:LO=LO+1:D$=D$+A$:IF LO<>15 THEN GOTO 5000
5060 PRINT SPC(16-LO)
5070 N$(NP)=D$
5080 HI=P(1)
5090 LOCATE 16,1:PRINT SPACE$(240)
5100 PU=0:NV=3
5110 IF NP<>0 THEN NP=0:GOSUB 5380:GOTO 4520
5120 RETURN
5130 REM
5140 REM *****************
5150 REM * INSTRUCCIONES *
5160 REM *****************
5170 REM
5180 COLOR ,,,,3
5190 LOCATE 17,4
5200 PRINT "COMETE TODOS LOS PUNTOS QUE PUEDAS"
5210 PRINT
5220 PRINT "    ^"
5230 PRINT "    Q          USA ESTAS TECLAS"
5240 PRINT "< O    P >"
5250 PRINT "    A       PARA MOVER EL QUESITO."
5260 PRINT "    v"
5270 RETURN
5280 REM
5290 REM *******************
5300 REM * PULSA UNA TECLA *
5310 REM *******************
5320 REM
5330 LOCATE 25,13
5340 PRINT "PULSA UNA TECLA"
5350 LET A$=INKEY$
5360 IF A$="" THEN GOTO 5350
5370 RETURN
5380 REM
5390 REM ****************
5400 REM * PRESENTACION *
5410 REM ****************
5420 REM
5430 CLS
5440 COLOR ,,,,3
5450 RESTORE 7070
5460 FOR I=2 TO 16
5470    READ A$
5480    FOR J=1 TO LEN(A$)
5490       IF MID$(A$,J,1)="0" THEN GOTO 5510
5500       PUT(67+8*(J-1),8*I),A1%
5510    NEXT J
```

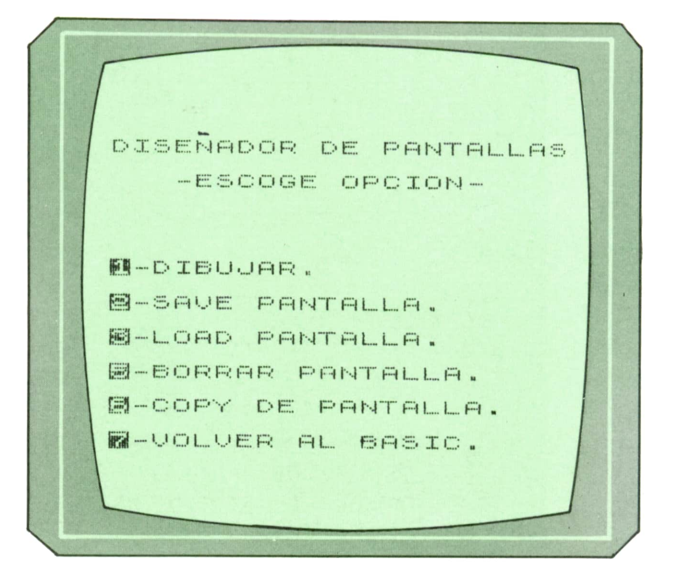
```
5520 NEXT I
5530 LOCATE 1,6:PRINT "(c) Ed. Siglo Cultural, 1987"
5540 COLOR ,,,,2
5550 FOR I=1 TO 14
5560    LOCATE 20,I:PRINT "."
5570    FOR J=1 TO 100:NEXT J
5580 NEXT I
5590 COLOR ,,,,3
5600 LOCATE 20,17:PRINT "= 10 PUNTOS"
5610 COLOR ,,,,2
5620 FOR I=1 TO 13
5630    LOCATE 22,I:PRINT " *"
5640    FOR J=1 TO 100:NEXT J
5650 NEXT I
5660 COLOR ,,,,3
5670 LOCATE 22,17:PRINT "= 50 PUNTOS"
5680 FOR J=1 TO 1000:NEXT J
5690 FOR I=0 TO 13
5700    PUT(I*8,152),A1%
5710    FOR J=1 TO 100:NEXT J
5720    LOCATE 20,I+1:PRINT " "
5730    PUT(I*8+4,152),B%
5740    FOR J=1 TO 100:NEXT J
5750    LOCATE 20,I+1:PRINT "  "
5760 NEXT I
5770 PUT(104,152),A1%
5780 LOCATE 20,17:PRINT "= TU MISMO "
5790 FOR I=0 TO 13
5800    PUT(I*8,168),C%
5810    FOR J=1 TO 100:NEXT J
5820    LOCATE 22,I+1:PRINT " "
5830    PUT(I*8+2,168),D%
5840    FOR J=1 TO 100:NEXT J
5850    LOCATE 22,I+1:PRINT "  "
5860 NEXT I
5870 PUT(104,168),C%
5880 LOCATE 22,17:PRINT "= TU ENEMIGO"
5890 FOR I=1 TO 1000
5900 NEXT I
5910 RETURN
5920 REM
5930 REM ***************************
5940 REM * DATAS CON LA DEFINICION *
5950 REM * DE LOS PERSONAJES.      *
5960 REM ***************************
5970 REM
5980 DATA 16,16,8,8,20485,20485,21525,21525,16469,21845,85,21845,85,21845
5990 DATA 16469,21845,21525,21525,20485,20485,64,0
6000 DATA 16,16,8,8,-24566,-24566,-22486,-22486,-30046,-30046,-32126,-30046
6010 DATA -22486,-22486,-24566,-24566,8200,10280,2080,640,0,0
6020 DATA 126,51,0,0,0,3840,-1,192,0,0,0,0,0,-241,-1,-16129,0,0,0,0,0,-1
6030 DATA -1,-769,0,0,0,0,-256,-1,-1,-1,252,0,0,0,-241,-1,-1,-1,-16129,0,0,0
6040 DATA -193,-1,-1,-1,-3841,0,0,0,-1,-1,-769,-241,-769,0,0,768,-1,-1,-16129
6050 DATA -256,-1,0,0,16128,-1,-1,4095,16380,-1,240,0,-256,-1,-1,4095,16188,-1
6060 DATA 252,0,-253,-1,-1,4095,16380,-1,255,0,-241,-1,-1,-16129,-256,-1
6070 DATA -16129,0,-241,-1,-1,-769,-241,-1,-16129,0,-1,-1,-1,-1,-1,-1,-769,0
6080 DATA -1,-1,-1,-1,-1,-1,-769,0,-1,-1,-1,-1,-1,-1,-16129,768,-1,-1,-1,-1
6090 DATA -1,-1,240,768,-1,-1,-1,-1,-1,-1,0,3840,-1,-1,-1,-1,-1,-16129,0,3840
6100 DATA -1,-1,-1,-1,-1,240,0,3840,-1,-1,-1,-1,-769,0,0,16128,-1,-1,-1,-1,255
6110 DATA 0,0,16128,-1,-1,-1,-1,240,0,0,16128,-1,-1,-1,-769,0,0,0,16128,-1,-1
6120 DATA -1,255,0,0,0,16128,-1,-1,-1,240,0,0,0,16128,-1,-1,-1,255,0,0,0
6130 DATA 16128,-1,-1,-1,-769,0,0,0,16128,-1,-1,-1,-1,192,0,0,16128,-1,-1,-1
6140 DATA -1,255,0,0,3840,-1,-1,-1,-1,-769,0,0,3840,-1,-1,-1,-1,-1,240,0
6150 DATA 3840,-1,-1,-1,-1,-1,-16129,0,768,-1,-1,-1,-1,-1,-769,0,768,-1,-1,-1
6160 DATA -1,-1,-1,240,0,-1,-1,-1,-1,-1,-1,-16129,0,-1,-1,-1,-1,-1,-1,-769
6170 DATA 0,-1,-1,-1,-1,-1,-1,-769,0,-241,-1,-1,-1,-1,-1,-16129,0,-241,-1,-1
6180 DATA -1,-1,-1,-16129,0,-253,-1,-1,-1,-1,-1,255,0,-256,-1,-1,-1,-1,-1,252
6190 DATA 0,16128,-1,-1,-1,-1,-1,240,0,768,-1,-1,-1,-1,-1,0,0,0,-1,-1,-1,-1
6200 DATA -769,0,0,0,-193,-1,-1,-1,-3841,0,0,0,-241,-1,-1,-1,-16129,0,0,0
```

```
6210 DATA -256,-1,-1,-1,252,0,0;0,0,-1,-1,-769,0,0,0,0,0,-241,-1,-16129
6220 DATA 0,0,0,0,0,3840,-1,192,0,0
6230 DATA 132,51,0,0,0,3840,-1,192,0,0,0,0,0,3840,-1,-1,192,0,0,0,0,0,-1,-1
6240 DATA -769,0,0,0,0,0,-1,-1,-1,-769,0,0,0,0,-241,-1,-1,-1,-16129,0,0,0
6250 DATA 16128,-1,-1,-1,-1,240,0,0,0,-1,-1,-1,-1,-769,0,0,0,-253,-1,-1
6260 DATA 4092,-1,255,0,0,16128,-1,-1,-16129,-256,-1,240,0,0,-1,-1,-1,-1009
6270 DATA -193,-769,0,0,-253,-1,-1,4095,16188,-1,255,0,3840,-1,-1,-1,-1009
6280 DATA -193,-1,192,0,-241,-1,-1,-16129,-256,-1,-16129,0,-256,-1,-1,-1
6290 DATA 4092,-1,-1,252,0,-1,-1,-1,-1,-1,-1,-769,0,-256,-1,-1,-1,-1,-1,-1
6300 DATA 192,768,-1,-1,-1,-1,-1,-1,12528,0,-253,-1,-1,-1,-1,-1,3324,15
6310 DATA 3840,-1,-1,-1,-1,-1,1023,-253,192,-241,-1,-1,-1,-1,-16129,-64
6320 DATA -16129,3840,-1,-1,-1,-1,-3841,16176,-1,192,-193,-1,-1,-1,-769
6330 DATA 3840,-1,-3841,16128,-1,-1,-1,-1,3840,-1,-1,240,-193,-1,-1,-1
6340 DATA 960,-1,-1,-3841,16128,-1,-1,-1,240,-1,-1,-1,240,-193,-1,-1,255
6350 DATA -193,-1,-1,-3841,16128,-1,-1,-1,-241,-1,-1,-1,240,-193,-1,-1,-1
6360 DATA -1,-1,-1,-3841,16128,-1,-1,-1,-1,-1,-1,-1,240,-193,-1,-1,-1,-1
6370 DATA -1,-1,-3841,3840,-1,-1,-1,-1,-1,-1,-1,192,-241,-1,-1,-1,-1,-1,-1
6380 DATA -16129,3840,-1,-1,-1,-1,-1,-1,-1,192,-253,-1,-1,-1,-1,-1,-1,255
6390 DATA 768,-1,-1,-1,-1,-1,-1,-1,0,-256,-1,-1,-1,-1,-1,-1,252,0,-1,-1,-1
6400 DATA -1,-1,-1,-769,0,-256,-1,-1,-1,-1,-1,-1,252,0,-241,-1,-1,-1,-1,-1
6410 DATA -16129,0
6420 DATA 3840,-1,-1,-1,-1,-1,-1,192,0,-253,-1,-1,-1,-1,-1,255,0,0,-1,-1
6430 DATA -1,-1,-1,-769,0,0,16128,-1,-1,-1,-1,-1,240,0,0,-253,-1,-1,-1,-1
6440 DATA 255,0,0,0,-1,-1,-1,-1,-769,0,0,0,16128,-1,-1,-1,-1,240,0,0,0
6450 DATA -241,-1,-1,-1,-16129,0,0,0,0,-1,-1,-1,-769,0,0,0,0,0,-1,-1,-769
6460 DATA 0,0,0,0,0,3840,-1,-1,192,0,0,0,0,0,3840,-1,192,0,0,0
6470 DATA 16,16,16,8,8,8,20485,20485,0,21525,21525,1040,21761,21845,1360
6480 DATA 21760,21845,5460,21760,5460,21845,21761,1360,21845,21525,1040,21525
6490 DATA 20485,0,20485,0,0,0
6500 REM
6510 REM ****************************
6520 REM * DATAS CON LOS CARACTERES *
6530 REM ****************************
6540 REM
6550 DATA 32,201,187,200,188,205,186,206,204,185
6560 REM
6570 REM *************************
6580 REM * DATAS CON LOS RECORDS *
6590 REM *************************
6600 REM
6610 DATA "ANTONIO",34990
6620 DATA "EL COCO",31720
6630 DATA "EL AUTOR",24030
6640 DATA "INDIANA",23300
6650 DATA "JOSELITO",11210
6660 DATA "SUSYN",11120
6670 DATA "RAMIRO",8430
6680 DATA "DONALD",7540
6690 DATA "SHE",3400
6700 DATA "HE",3210
6710 REM
6720 REM *****************************
6730 REM * DATAS CON LAS DIRECCIONES *
6740 REM *****************************
6750 REM
6760 DATA -1,-1,1,1,0,0,0,0,0,0,0,0,-1,-1,1,1
6770 REM
6780 REM ******************************************
6790 REM * DATAS CON LA DEFINICION DE LA PANTALLA *
6800 REM ******************************************
6810 REM
6820 DATA "0015555555555555552000001555555555555200"
6830 DATA "0060000000000000006000006000000000000600"
6840 DATA "15A555555555555B5A5555B5A555555555555A52"
6850 DATA "6*..............35555554..............*6"
6860 DATA "6.15552.1555552..........1555552.15552.6"
6870 DATA "6.60006.600000352.1552.154000006.60006.6"
6880 DATA "6.35554.355555554.6006.355555554.35554.6"
6890 DATA "6.................6006.................6"
```

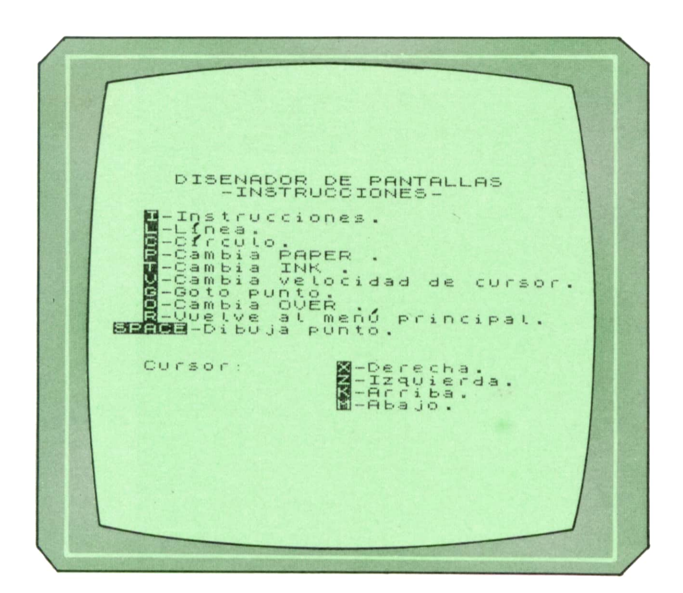
```
6900 DATA "8555552.155552.1554003552.155552.1555559"
6910 DATA "6155526.601554.3555555554.355206.6155526"
6920 DATA "6355546.606..................606.6355546"
6930 DATA "3555554.354.152.1CCCCCC2.152.354.3555554"
6940 DATA "............606.60000006.606............"
6950 DATA "15555B2.152.606.60000006.606.152.1B55552"
6960 DATA "6....34.354.354.35555554.354.354.34....6"
6970 DATA "6.12................................12.6"
6980 DATA "6.63552.15555555555555555555552.15546.6"
6990 DATA "6.35526.60001555555555555555520006.61554.6"
7000 DATA "6....34.35554..............35554.34....6"
7010 DATA "6.12..........15B555555B52..........12.6"
7020 DATA "6.34.15555552.354......354.15555552.34.6"
7030 DATA "6*...35555206.....1552.....60155554...*6"
7040 DATA "8555555555A5755555A55A5555BA5A555555559"
7050 DATA "6000000000006000000000000006000000000006"
7060 DATA "3555555555554000000000000003555555555554"
7070 REM
7080 REM ********************************
7090 REM * DATAS CON LA PALABRA PAC-MAN *
7100 REM ********************************
7110 REM
7120 DATA "111111000011000001111 0"
7130 DATA "0110011001111000110011"
7140 DATA "0110011011001101100000"
7150 DATA "0111100011001101100000"
7160 DATA "0110000011111101100000"
7170 DATA "0110000011001100110011"
7180 DATA "1111000011001100011110"
7190 DATA ""
7200 DATA "11000110000110001100011"
7210 DATA "11101110001111001110011"
7220 DATA "11111110011001101111011"
7230 DATA "11111110011001101101111"
7240 DATA "11010110011111101100111"
7250 DATA "11000110011001101100011"
7260 DATA "11000110011001101100011"
```

Programa: Diseñador de pantallas para SPECTRUM

La utilidad que presentamos a continuación es válida únicamente para el SPECTRUM. El programa nos permitirá definir nuestras propias pantallas gráficas y almacenarlas en cassette para cargarlas más tarde con un simple LOAD "" CODE.




Menú de opciones.



El programa lleva en sí mismo todas las instrucciones necesarias para que el usuario entienda su funcionamiento. Para ello, una vez que se encuentre en la pantalla de dibujo, sólo tiene que pulsar la tecla I de INSTRUCCIONES.


Ilustraciones.

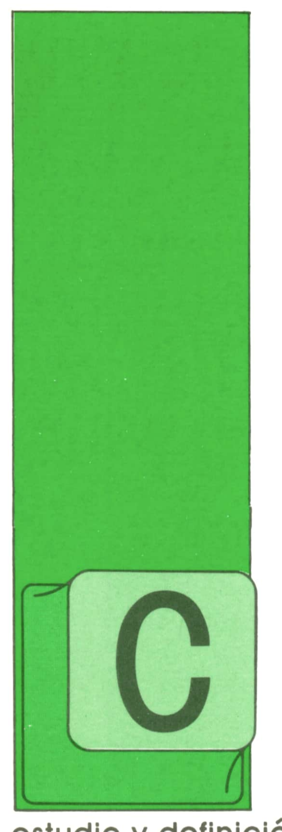
```
  1 REM ********************************
  2 REM * (c) Ediciones Siglo Cultural *
  3 REM * (c) 1987                     *
  4 REM ********************************
  5 REM
  8 PAPER 0: BORDER 0: INK 7: CLEAR 48899: GO SUB 9800
  9 CLS : RANDOMIZE USR 48900
 10 CLS
 11 LET x=128
 12 LET y=88
 13 LET d=1
 14 LET o=1
 15 LET p=0
 16 LET tin=7
 17 LET pap=0
 18 GO TO 9500
 19 REM
 20 REM ****************************************
 25 REM * SITUA EL CURSOR EN SU NUEVA POSICION *
 26 REM ****************************************
 27 REM
 30 PLOT  PAPER PAP;X,Y
 40 LET X=X+(D AND I$="X" AND D+X<256)-(D AND I$="Z" AND X-D>-1): LET Y=Y+(D AN
D I$="K" AND Y+D<176)-(D AND I$="M" AND Y-D>-1)
 55 PLOT  PAPER PAP;X,Y
 60 INPUT "": PRINT #0;"X=";X;"  ";TAB 7;"Y=";Y;"  ","OVER:";O
 70 RETURN
 80 REM
 90 REM ****************************
100 REM * DIBUJO PROPIAMENTE DICHO *
101 REM ****************************
102 REM
110 LET i$=INKEY$
115 IF I$="" THEN  GO TO 110
120 IF I$="Z" OR I$="X" OR I$="K" OR I$="M" THEN  GO SUB 20
130 IF i$="O" THEN  LET O=(O=0): OVER O: PLOT  OVER 1;X,Y: GO SUB 60
140 IF I$="I" THEN  GO SUB 9000
150 IF I$="P" THEN  INPUT "PAPER ? ";T: IF T>-1 AND T<8 THEN  LET PAP=T: PAPER
PAP
160 IF I$="T" THEN  INPUT "TINTA ? ";T: IF T>-1 AND T<8 THEN  LET TIN=T: INK T
170 IF I$="G" THEN  INPUT "GOTO X=";T1;"  Y=";T2: IF FN H(T1) AND FN V(T2) THEN
 PLOT  OVER 1;X,Y: LET X=T1: LET Y=T2: PLOT  OVER 1;X,Y: GO SUB 60
180 IF I$="L" THEN  PLOT X1,Y1: DRAW X-X1,Y-Y1
```

```
 190 IF I$="C" THEN  INPUT "RADIO ? ";T: IF FN H(T+X) AND FN V(T+Y) THEN  CIRCLE
 X,Y,T
 200 IF I$="V" THEN  INPUT "VELOCIDAD DEL CURSOR ? ";T: IF FN V(T) THEN  LET D=T
 210 IF I$="R" THEN  RANDOMIZE USR 48900: RETURN
 220 IF I$=" " THEN  PLOT  OVER 1;X,Y: PAUSE 2: LET X1=X: LET Y1=Y: PAUSE 2: PAU
SE 2
 800 GO TO 110
8998 REM
8999 REM *****************
9000 REM * INSTRUCCIONES *
9001 REM *****************
9002 REM
9010 RANDOMIZE USR 48900
9011 INK 0
9012 PAPER 7
9013 OVER 0
9015 DEF FN c$(a,a$)=CHR$ 22+CHR$ a+CHR$ INT ((32-LEN a$)/2)+a$
9016 DEF FN H(T)=(T>-1 AND T<256): DEF FN V(T)=(T>-1 AND T<176)
9020 CLS
9021 PRINT FN C$(0,"DISENADOR DE PANTALLAS"),FN C$(1,"-INSTRUCCIONES-")
9025 LET t=3
9026 PRINT
9030 PRINT TAB t; INVERSE 1;"I"; INVERSE 0;"-Instrucciones."
9040 PRINT TAB t; INVERSE 1;"L"; INVERSE 0;"-Linea."
9050 PRINT TAB t; INVERSE 1;"C"; INVERSE 0;"-Circulo."
9060 PRINT TAB t; INVERSE 1;"P"; INVERSE 0;"-Cambia  PAPER ."
9070 PRINT TAB t; INVERSE 1;"T"; INVERSE 0;"-Cambia INK ."
9080 PRINT TAB t; INVERSE 1;"V"; INVERSE 0;"-Cambia velocidad de cursor."
9090 PRINT TAB t; INVERSE 1;"G"; INVERSE 0;"-Goto punto."
9105 PRINT TAB T; INVERSE 1;"O"; INVERSE 0;"-Cambia OVER ."
9110 PRINT TAB t; INVERSE 1;"R"; INVERSE 0;"-Vuelve al menu principal."
9112 PRINT " "; INVERSE 1;"SPACE"; INVERSE 0;"-Dibuja punto."
9115 PRINT ''TAB T;"Cursor:", INVERSE 1;"X"; INVERSE 0;"-Derecha.",, INVERSE 1;"
Z"; INVERSE 0;"-Izquierda.",, INVERSE 1;"K"; INVERSE 0;"-Arriba.",, INVERSE 1;"M
"; INVERSE 0;"-Abajo."
9120 PRINT #0;FN c$(0,"-PULSA UNA TECLA-")
9130 PAUSE 0
9131 INK TIN
9132 PAPER PAP
9133 OVER 0
9140 RANDOMIZE USR 48912
9150 RETURN
9498 REM
9499 REM ******************
9500 REM * MENU PRINCIPAL *
9501 REM ******************
9502 REM
9520 POKE 23658,8
9521 INK 0
9522 PAPER 7
9523 CLS
9524 PRINT FN C$(0,"DISENADOR DE PANTALLAS")
9530 PRINT FN C$(2,"-ESCOGE OPCION-")
9540 PRINT : PRINT '''TAB 5; INVERSE 1;"1"; INVERSE 0;"-DIBUJAR."''TAB 5; INVERS
E 1;"2"; INVERSE 0;"-SAVE PANTALLA."''TAB 5; INVERSE 1;"3"; INVERSE 0;"-LOAD PAN
TALLA."''TAB 5; INVERSE 1;"5"; INVERSE 0;"-BORRAR PANTALLA."''TAB 5; INVERSE 1;"
6"; INVERSE 0;"-COPY DE PANTALLA."''TAB 5; INVERSE 1;"7"; INVERSE 0;"-VOLVER AL
BASIC."
9550 INPUT OPC
9560 IF OPC=1 THEN  INK TIN: PAPER PAP: OVER 0: RANDOMIZE USR 48912: PLOT  OVER
1;X,Y: GO SUB 60: GO SUB 100: GO TO 9500
9570 IF OPC=2 THEN  GO SUB 9630: GO TO 9520
9580 IF OPC=3 THEN  GO SUB 9660: GO TO 9520
9590 IF OPC=7 THEN  STOP
9600 IF OPC=5 THEN  CLS : RANDOMIZE USR 48900: GO TO 9500
9610 IF OPC=6 THEN  RANDOMIZE USR 48912: COPY
9620 GO TO 9500
9621 REM
9622 REM *****************
```

```
9630 REM * SAVE PANTALLA *
9631 REM *****************
9632 REM
9640 GO SUB 9700
9642 IF LEN N$>10 OR N$="" THEN  RETURN
9650 RANDOMIZE USR 48912
9655 SAVE N$SCREEN$
9657 RETURN
9658 REM
9659 REM *****************
9660 REM * LOAD PANTALLA *
9661 REM *****************
9662 REM
9664 GO SUB 9700
9665 IF LEN N$>10 THEN  RETURN
9670 LOAD N$SCREEN$
9671 INPUT ""
9672 RANDOMIZE USR 48900
9680 RETURN
9700 INPUT "NOMBRE ? (MAX. 10 CARACTERES)"' LINE N$
9710 RETURN
9798 REM
9799 REM ***************
9800 REM * RUTINAS C/M *
9801 REM ***************
9802 REM
9820 FOR I=48900 TO 48923
9821 READ A
9822 POKE I,A
9823 NEXT I
9830 RETURN
9840 REM
9841 REM ******************
9842 REM * LINEAS DE DATA *
9843 REM ******************
9844 REM
9850 DATA 33,0,64,17,80,195,1,0,27,237,176,201,33,80,195,17,0,64,1,0,27,237,176,
201
```

TECNICAS DE ANALISIS

FASES DE LOS ESTUDIOS INFORMATICOS



COMO se ha comentado, el campo de aplicación natural del «esquema director» son los diferentes «dominios» de la actividad de la Compañía. Es precisamente el estudio y definición de estos dominios lo que aporta la información imprescindible para el establecimiento de los «escenarios» generales sobre los cuales se desarrolla la actividad de la Organización.

Tal como hemos descrito, se suelen establecer fundamentalmente tres tipos de estudios de base para el desarrollo de los proyectos informáticos, según el nivel de mecanización de la Compañía o Departamento de que se trate y según la amplitud del proyecto que se aborda: el esquema director, el estudio de viabilidad y el estudio detallado. Veamos las etapas que, en el desarrollo de estos estudios, suelen aparecer.



Esquema director

Suele desarrollarse en cinco fases:

— **Estudio de la puesta en marcha** del plan. Se trata de comprobar que se dan las condiciones objetivas necesarias para el lanzamiento de un estudio de este tipo: definición de problemas de gran amplitud, interés por parte de la dirección, estado adecuado del nivel de mecanización y del desarrollo de los problemas, etc. Se establecen las grandes decisiones a tomar: descentralización-concentración, fenómeno micro, comunicaciones, etc.

— **Balance de la situación y objetivos a alcanzar.**

Se establece un balance exhaustivo por dominios verticales (producción, comercial, etc.) y por temas horizontales (comunicación, ayuda a la decisión, etcétera). Se trata de obtener una primera aproximación a los objetivos a alcanzar, a los esfuerzos futuros que habrá que aportar, y relacionar todo ello con la situación de que se parte. Al final de esta fase se pueden apuntar algunas soluciones conceptuales: recentralizar tal tipo de información para obtener una mayor coordinación, controlar más de cerca ciertas funciones para establecer, posiblemente, una nueva distribución de tareas, etc.

— **Concepción de los escenarios posibles.**

Un escenario se define por una solución organizacional y técnica y por una secuencia de tareas que hay que desarrollar para llegar a ella, teniendo en cuenta la situación de que partimos y las posibilidades de evolución.

Cuando se concluye el estudio de los escenarios posibles se pueden comparar las diferentes variaciones elaboradas, desde el punto de vista de los costes, riesgos y ventajas de cada una y establecer un cuadro de soluciones propuestas, así como el orden de magnitud de las inversiones necesarias para desarrollarlas.

— **Elaboración de planes de acción.**

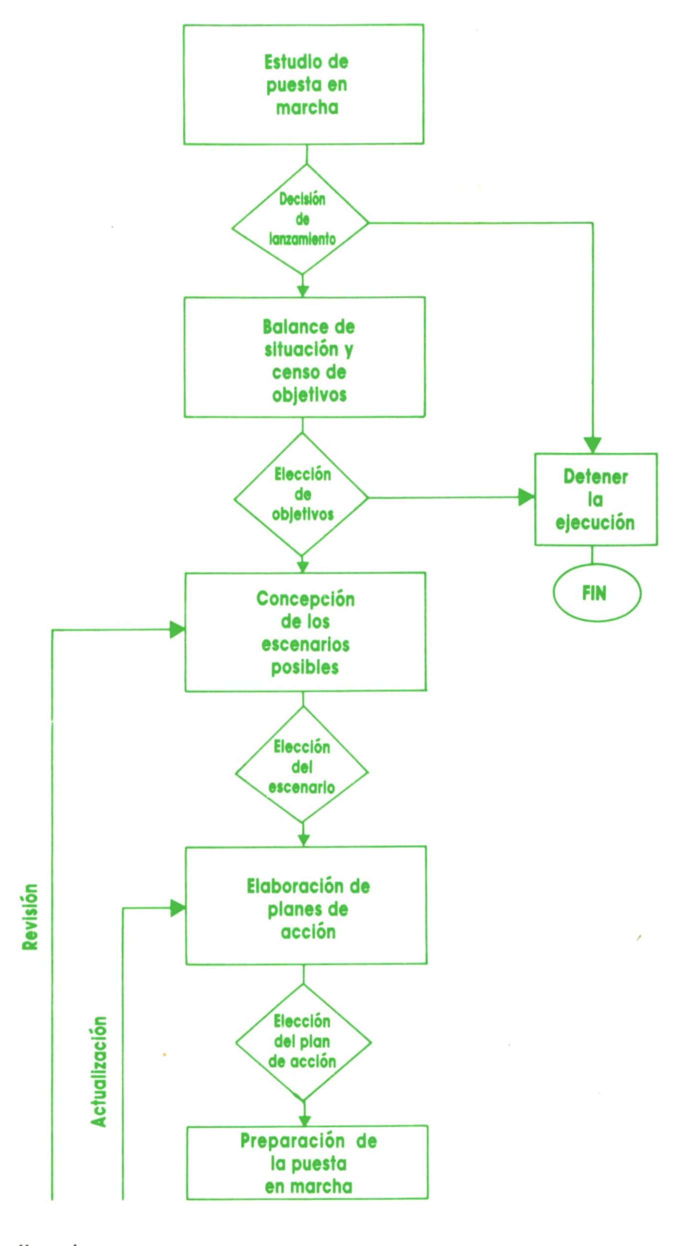
Se trata de ajustar un «planning» de las acciones a abordar en los próximos años

en función de los medios de que se dispone.

Hay que relacionar los proyectos que se han de poner en marcha, los materiales que es necesario adquirir, etc. Es importante establecer las prioridades de cada tarea, su desarrollo en el tiempo y la condición de modificable (actualizable) o no de cada uno de los planes propuestos.

— **Preparación de la puesta en marcha** del plan de acción que se haya elegido. La puesta en marcha de un plan de acción supone la definición de estructuras y procedimientos para:

- realizar lo que se ha previsto
- estar preparado para abordar los imprevistos

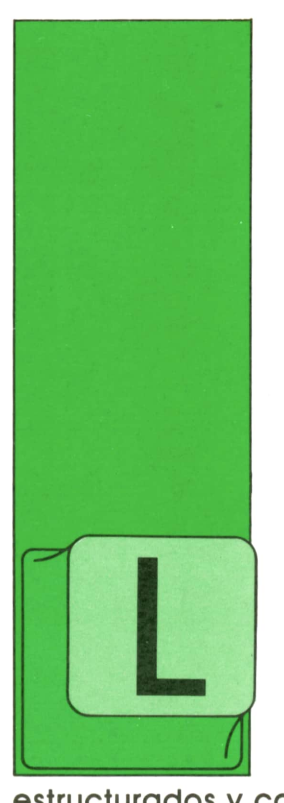



Ciclo del «esquema director».

- mantener un sistema de actualización periódica del plan de acción.

Como conclusión de esta parte del trabajo conviene reflexionar sobre el estudio completo, validar los planes propuestos y decidir la aprobación final del documento «esquema director» para, consecuentemente, «lanzarlo» a la organización: difundirlo y estudiarlo.

TECNICAS DE PROGRAMACION





Programación modular

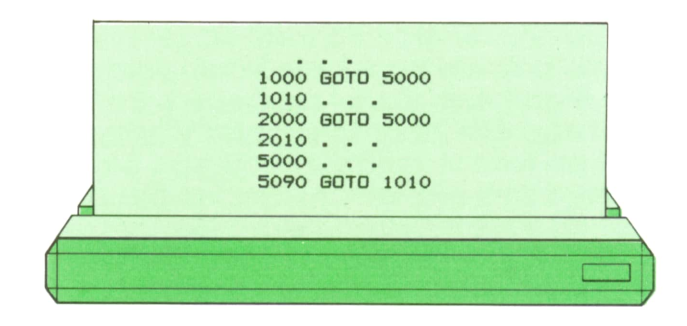
La programación modular es una de las técnicas más importantes y útiles que se utilizan en programación de ordenadores. Su uso permite hacer los programas más estructurados y comprensibles, más fáciles de analizar y construir, facilitando, además, el trabajo en equipo, pues un programa completo puede descomponerse en módulos hasta cierto punto independientes, cada uno de los cuales puede ser asignado a un programador diferente, permitiendo al mismo tiempo a cada miembro del equipo cierta libertad para programar y elegir nombres de variables de acuerdo con su estilo personal.

Desgraciadamente, el lenguaje BASIC, uno de los más extendidos en el campo de los microordenadores, aquí, como en otras cosas, deja mucho que desear. En efecto, este lenguaje no está realmente preparado para la programación modular. Existen, es verdad, ciertas instrucciones que permiten programar de una forma que se aproxima lejanamente a la utilización de estas técnicas, pero que, comparadas con las que ofrecen otros lenguajes, resultan insuficientes e insatisfactorias.

A pesar de todo, y para quien no tenga otra posibilidad que la de utilizar el lenguaje BASIC, es preciso conocer con detalle lo poco que puede hacerse, para aprovecharlo al máximo en lo que pueda dar de sí y sacar alguna ventaja, por pequeña que sea, de la programación modular.

Llamaremos «módulo» (o subprograma) de un programa a una parte aislada de éste, cuya ejecución puede desencadenarse desde diversos puntos del resto del programa. Además (y esto es lo fundamental), una vez que termine la ejecución del módulo, el programa debe continuar ejecutándose en la instrucción siguiente a aquella que invocó la ejecución del módulo.

Veamos, en primer lugar, un ejemplo de lo que no es programación modular:



```
. . .
1000 GOTO 5000
1010 . . .
2000 GOTO 5000
2010 . . .
5000 . . .
5090 GOTO 1010
```

Observemos lo que ocurre con este programa. En primer lugar, cuando quiera que la ejecución llegue a la instrucción 1000, la instrucción de transferencia incondicional transfiere control a la instrucción 5000. Allí se ejecuta una serie de instrucciones (representadas en el ejemplo por puntos suspensivos) después de lo cual se transfiere de nuevo control a la instrucción 1.010 y se continúa la ejecución secuencial de las instrucciones sucesivas. Es decir, el trozo de programa comprendido entre las instrucciones 5000 y 5090 se ha insertado, para todos los efectos, entre las instrucciones 1000 y 1010. Hasta aquí todo está de acuerdo con la definición de la programación modular indicada más arriba.

Sin embargo, la situación es muy diferente si llegamos a ejecutar la instrucción 2000. En ese caso, como en el anterior, nos encontramos una transferencia in-

condicional a la instrucción 5000. Allí se ejecutará de nuevo la serie de instrucciones de etiquetas 5000 a 5090, y al llegar a esta última daremos con una nueva instrucción de transferencia incondicional, que nos envía de nuevo a la instrucción 1010. No se cumple, por tanto, la condición fundamental de la programación modular: que al final de la ejecución del módulo, el programa continúe en la instrucción siguiente a la que puso en marcha la ejecución de aquél. En nuestro caso, para que la programación fuese modular, en este segundo caso la ejecución tendría que haber regresado a la instrucción 2010.

En BASIC existe una instrucción que permite desencadenar la ejecución del módulo y que se escribe así:

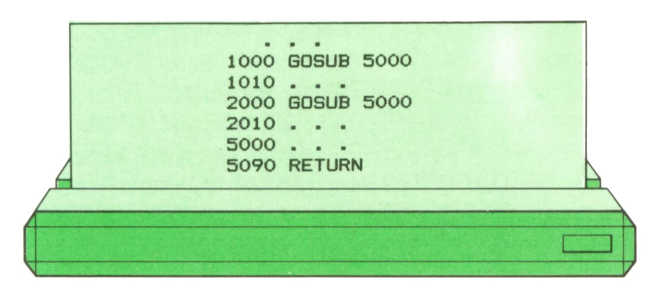
GOSUB etiqueta

Además, se dispone también de una instrucción especial que indica al intérprete o compilador de BASIC que se da por terminada la ejecución del módulo y se desea regresar a la instrucción siguiente a la que desencadenó dicha ejecución. Dicha instrucción de terminación se escribe así:

RETURN

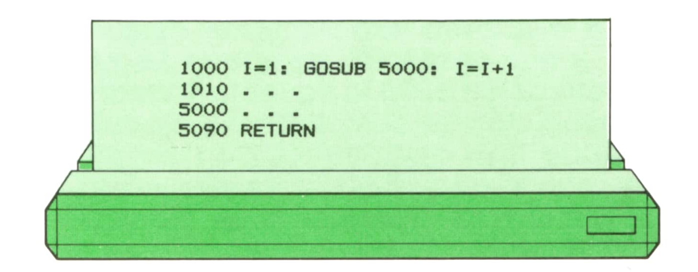
que, en inglés, significa VOLVER.

Veamos un ejemplo de programación modular:

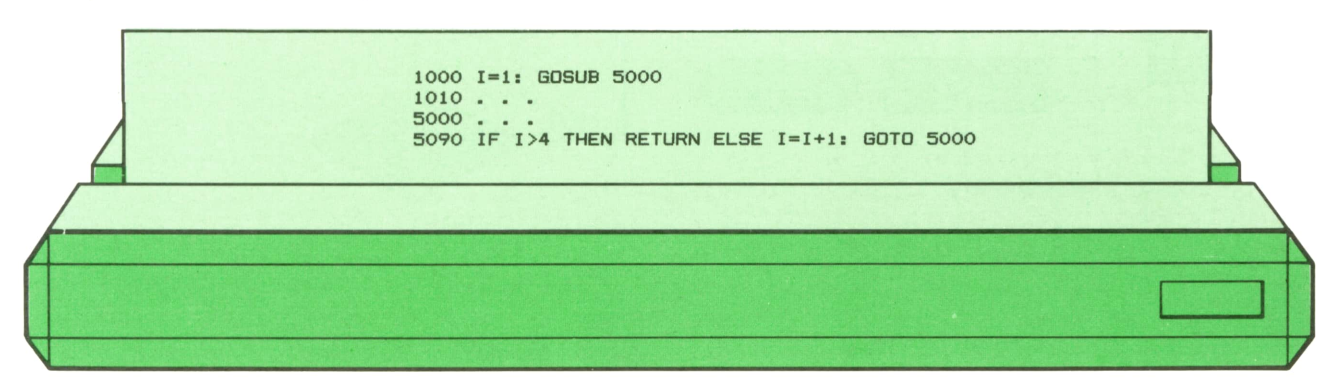


Este caso se diferencia del anterior en lo siguiente: el paso por la instrucción 1000 desencadena la ejecución del módulo comprendido entre las instrucciones 5000 y 5090. Al llegar a ésta (RETURN) la ejecución vuelve a la instrucción 1010. Hasta aquí, todo va como en el ejemplo anterior. Pero si ahora llegamos a la instrucción 2000, se desencadenará de nuevo la ejecución de la instrucción 5000, y al llegar por segunda vez a la instrucción 5090, el programa regresará a la instrucción 2010. Es decir, en la programación modular, el programa «recuerda» el lugar desde donde ha sido llamado y es capaz de volver a él cuando encuentra la instrucción RETURN.

Tanto la instrucción GOSUB como RETURN pueden emplearse dentro de un bloque secuencial de instrucciones, o como parte ejecutable de una instrucción condicional, o dentro de un bucle. Veamos algunos ejemplos:



La ejecución de la instrucción 1000 se realiza así: primero se asigna a la variable I el valor 1. Después se desencadena la ejecución del módulo situado en la etiqueta 5000. Cuando éste llega a su fin (la instrucción de etiqueta 5090) y encuentra la correspondiente instrucción RETURN, la ejecución del programa principal continúa en el punto donde se había interrumpido, es decir, se le asigna a la variable I el valor I + 1. Entonces, y sólo entonces, pasa la ejecución a la instrucción 1010.



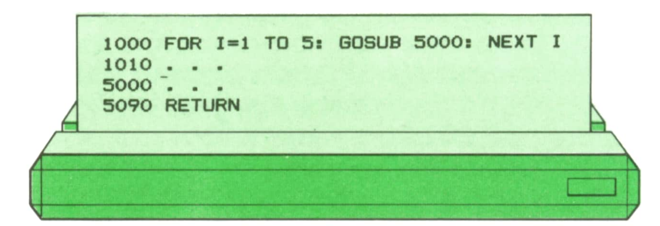
En este ejemplo, la instrucción 1000 asigna a la variable I el valor 1 y después desencadena la ejecución del módulo que comienza en la etiqueta 5000. Este se ejecuta sucesivamente hasta llegar a la instrucción 5090, donde encontramos una instrucción condicional cuya condición (I > 4) no se cumple (suponemos que el valor de I no es modificado por las instrucciones del módulo anteriores a la etiqueta 5090). Por tanto, entra en ejecución la parte ELSE, que consta de un bloque secuencial de dos instrucciones. La primera incrementa el valor de I en una unidad (con lo que pasa a valer 2) y la segunda es una transferencia incondicional al principio del módulo. La ejecución de éste se repetirá, por tanto, pero ahora, al llegar por segunda vez a la instrucción 5090, la variable I vale 2. Tampoco se cumple la condición, por lo que asignaremos a I el valor 3 y volveremos a ejecutar el módulo. Es fácil ver que el módulo se ejecutará exactamente cinco veces, y que la quinta vez que se llegue a la instrucción 5090 la variable 1 valdrá 5, la condición se cumplirá y se ejecutará la parte THEN de la instrucción condicional, con lo que se dará por terminada la ejecución del módulo y la ejecución continuará en la instrucción de etiqueta 1010.

Veamos cómo sería el organigrama del ejemplo anterior:

Fig. 1.

Obsérvese que el organigrama de un módulo se dibuja como un programa independiente, que la instrucción RETURN corresponde al bloque curvo que indica el final del programa y que el desencadenamiento de la ejecución del módulo se representa con un bloque rectangular al que, para hacerlo resaltar, se le realza a menudo con dos líneas verticales adicionales.

Veamos otro ejemplo:



```
1000 FOR I=1 TO 5: GOSUB 5000: NEXT I
1010 . . .
5000 . . .
5090 RETURN
```

En este caso, la llamada a la ejecución del módulo (GOSUB 5000) aparece dentro de una instrucción de bucle. En las condiciones indicadas, el módulo se ejecutará exactamente cinco veces. Por tanto, si las instrucciones del módulo representadas por puntos suspensivos (5000 a 5080) fueran las mismas que las del ejemplo anterior, este programa sería totalmente equivalente al de dicho ejemplo.

Veamos su organigrama:

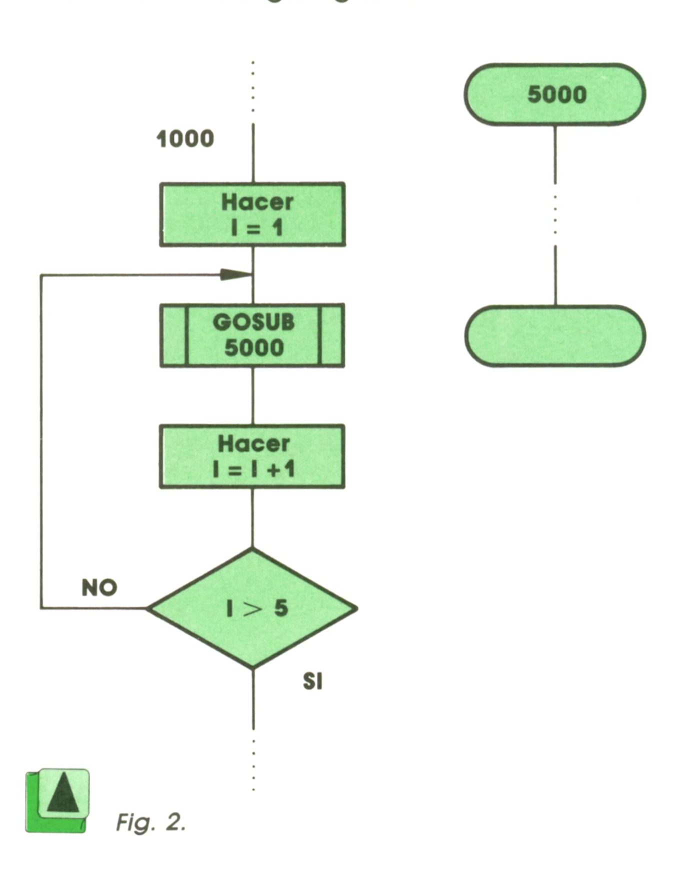
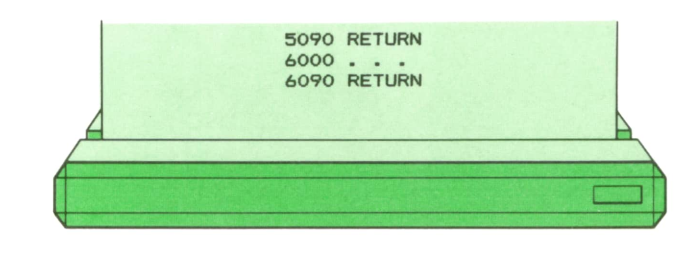


Fig. 2.

Citemos, por último, que es también posible llamar a un módulo dentro de otro módulo, como en el siguiente ejemplo:

```
1000 GOSUB 5000
1010 . . .
5000 . . .
5050 GOSUB 6000
5060 . . .
```



cuyo organigrama es:

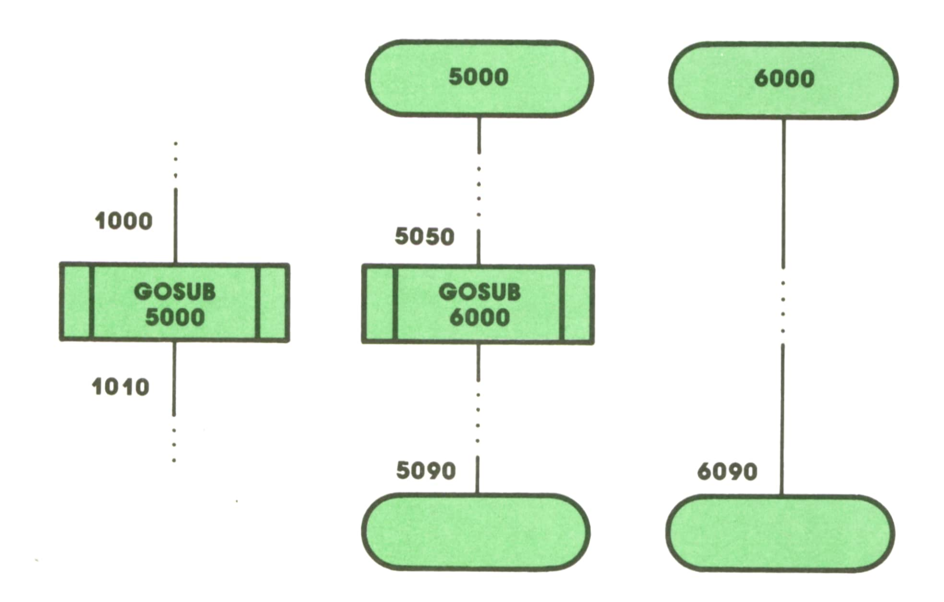
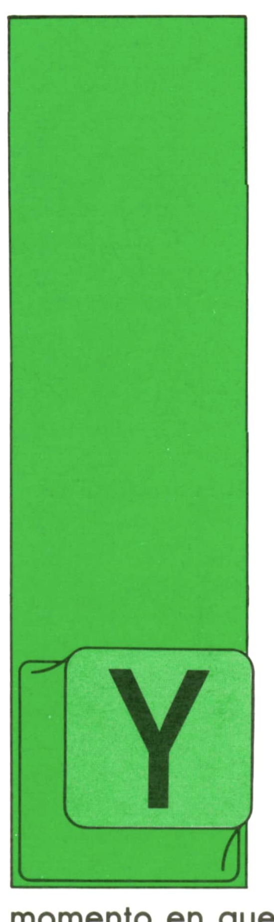



Fig. 3.

En este caso, al llegar a la instrucción 1000, se desencadena la ejecución del módulo que comienza en la instrucción 5000. A medida que ésta avanza, se llega a la instrucción 5050, donde se desencadena la ejecución del módulo que comienza en la instrucción 6000. Al llegar a la instrucción 6090, se da por terminada la ejecución de este módulo y regresamos a la instrucción siguiente a la que lo llamó (en nuestro caso, la instrucción 5060). Después continuamos sucesivamente hasta llegar a la instrucción 5090, donde termina, a su vez, la ejecución del módulo que comienza en la instrucción 5000, regresándose, por fin, a la instrucción 1010. Como se ve, el intérprete o el compilador de BASIC dispone de una pila donde guardan las direcciones de retorno de cada módulo, con el fin de regresar cada uno a la instrucción que le corresponde, aunque sean varias las llamadas acumuladas.

LOGO

Cómo copiar los procedimientos y variables en una cinta o diskette



A sabemos que tenemos la posibilidad de enseñarle a la tortuga a hacer cosas nuevas mediante la definición de procedimientos. Pero hay un problema. En el momento en que apagamos el ordenador, la tortuga se olvida de todo lo que le hemos enseñado y si queremos que lo vuelva a aprender, tenemos que definir otra vez los procedimientos.

Para evitar esto, lo que se puede hacer es guardar todo lo que ha aprendido la tortuga en una cinta o en un diskette antes de apagar el ordenador. De esta forma, cuando volvamos a encenderlo no hará falta escribir los procedimientos otra vez, sino que bastará con darle una orden a la tortuga para que lea lo que tenemos almacenado en la cinta o diskette y lo aprenda de nuevo.

En la cinta o diskette no sólo podemos guardar la definición de procedimientos, sino que también se pueden almacenar los nombres de las variables a las que hayamos asignado un valor mediante el comando HAZ y los valores que tengan sus cajones correspondientes. Toda esta información se guarda en un *archivo* o *fichero.*

Es decir, es como si copiáramos todas aquellas cosas que le hemos enseñado a la tortuga y los cajones que hemos usado en una hoja de papel, pero en lugar de utilizar un cuaderno cogemos una cinta o un diskette y cada hoja es un fichero.

Para realizar esta copia se utiliza el comando

GUARDA "nombrefich "nombreproc

que copia la definición del procedimiento *nombreproc* en un fichero que vamos a llamar *nombrefich.* Como se puede ver, al igual que con los procedimientos, para distinguir un fichero de otro tenemos que dar a cada uno un nombre.

Si queremos copiar varios procedimientos en un mismo fichero, pondremos:

GUARDA "nombrefich [nombreproc1
nombreproc2 ... nombreprocn]

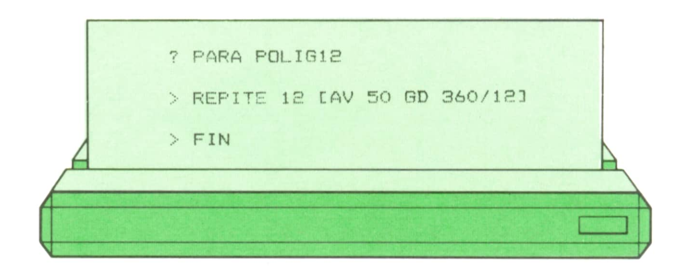
es decir, meteremos los nombres de los procedimientos en una lista.

Por último, si escribimos simplemente

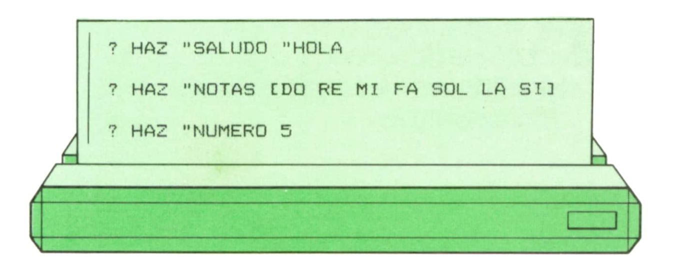
GUARDA "nombrefich

lo que haremos será copiar en el fichero *nombrefich* todas las definiciones de procedimientos que tengamos y, además, todos los nombres de variables que hayamos usado con el comando HAZ, así como los valores que tienen en ese momento.

Por ejemplo, supongamos que le hemos enseñado a la tortuga un procedimiento que dibuja un polígono de 12 lados:



y que luego hemos hecho varias asignaciones:



Si ahora guardamos todo en un fichero:

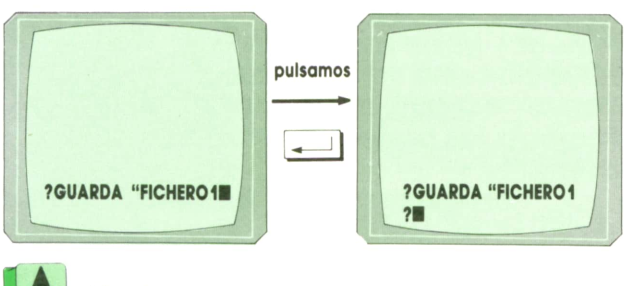


Fig. 1.

el fichero FICHERO1 contendrá:

— Las definiciones de procedimientos: POLIG12.
— Los nombres de las variables: SALUDO, NOTAS y NUMERO.
— Los valores que contienen estas variables: HOLA, DO RE MI FA SOL LA SI y 5.

Como vemos, al hacer la copia, la tortuga no cambia ni sus características ni lo que haya en la pantalla. Simplemente, ejecuta la orden que le hemos dado y nos indica que ha realizado el comando GUARDA escribiendo la interrogación (?), en espera de que le demos otra orden.

Si ya hemos terminado de trabajar, podemos apagar el ordenador sin importarnos que la totuga se olvide de las cosas nuevas que ha aprendido, ya que lo tenemos guardado en la cinta o diskette.

Cómo recuperar lo que hay guardado en la cinta o diskette

Cuando encendemos el ordenador y queremos enseñarle a la tortuga cosas que tenemos guardadas en una cinta o diskette, ya sabemos que no hace falta escribir los procedimientos otra vez. Pero nos falta saber cómo le podemos decir que lea las definiciones que están almacenadas en un fichero y las aprenda.

Para eso se utiliza el comando

CARGA «nombrefich

donde *nombrefich* es el nombre del fichero que contiene los procedimientos que la tortuga ha de aprender.

Vamos a ver si funciona. Supongamos que tenemos definidos varios procedimientos. Para ver cuáles son podemos utilizar el comando IMTS que nos muestra en la pantalla todos los nombres de procedimientos:

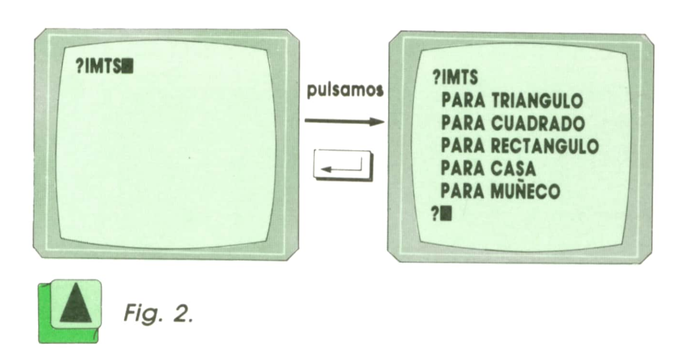
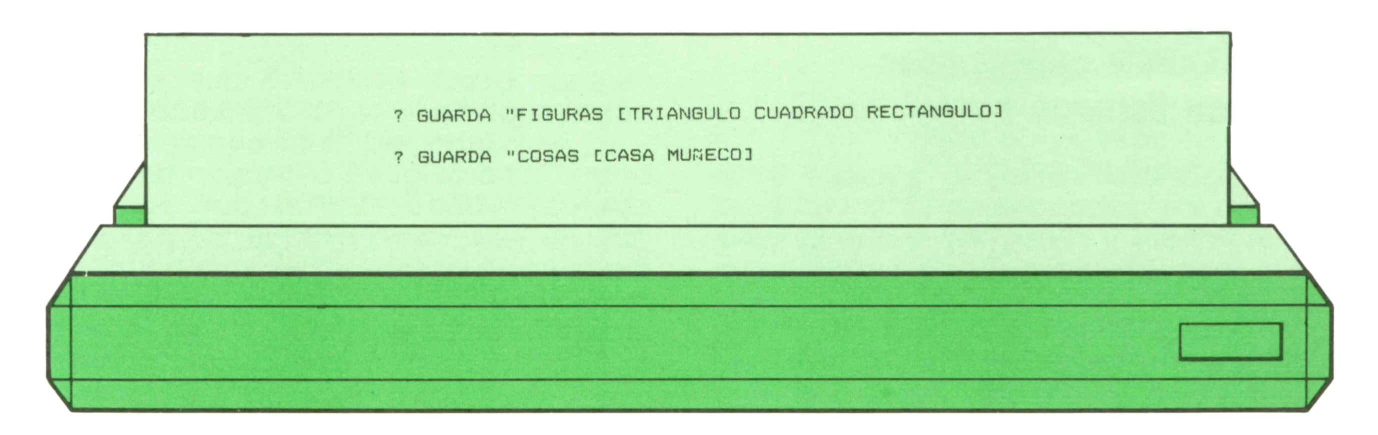


Fig. 2.

Ahora queremos guardarlos en dos ficheros diferentes: uno que contenga figuras y otro que contenga cosas. Para ello, escribiríamos:



Si ahora apagamos el ordenador y volvemos a encenderlo, al escribir IMTS nos saldrá:

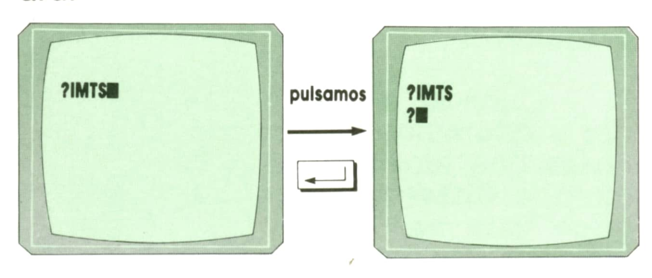



Fig. 3.

es decir, no hay ningún nombre de procedimiento (la tortuga ha olvidado todo lo que había aprendido).

Si queremos que sólo recuerde cómo hacer las figuras pondríamos:



y al escribir IMTS nos aparece lo siguiente:

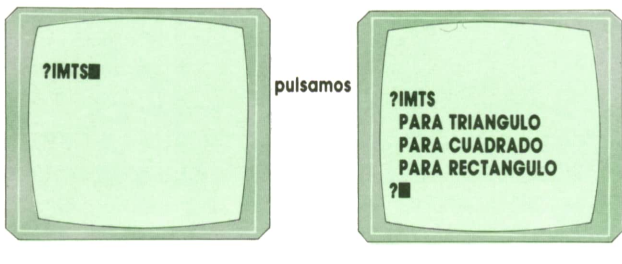



Fig. 4.

es decir, ya sabe hacer los dibujos correspondientes a los procedimientos contenidos en el archivo FIGURAS.

Algunas aplicaciones con ficheros

Ya sabemos cómo guardar información de procedimientos y variables en un fichero y cómo hacer que la tortuga la recupere después de apagar el ordenador.

Puede ocurrir que poco a poco hayamos ido creando muchos ficheros en nuestra cinta o diskette y que a la hora de que queramos que la tortuga aprenda los procedimientos almacenados en uno de ellos no recordemos su nombre.

El comando

DIR

(que es la abreviatura de DIRectorio) hace que la tortuga nos muestre en pantalla todos los nombres de los ficheros de la cinta o diskette. Es probable que estos nombres no sean exactamente los que nosotros pusimos, sino que lleven a continuación una serie de caracteres que se llaman en conjunto *extensión.* Estos caracteres los pone el ordenador automáticamente.

Por ejemplo, en el caso anterior tendríamos que:

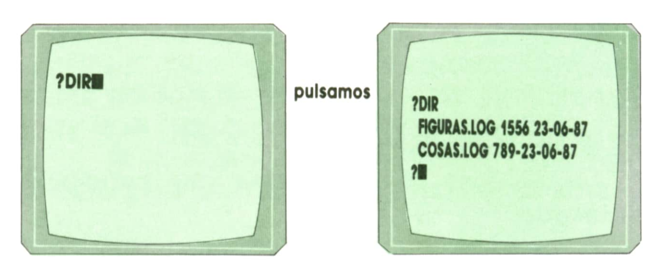



Fig. 5.

Aquí, la extensión es .LOG.

Es decir, con DIR obtenemos una lista de los ficheros que hay en la cinta o diskette. Es parecido al índice de un libro, pero en lugar de tener los títulos de los temas y las páginas, nos aparecen los nombres de los ficheros y otra información, como su tamaño, su fecha de creación...

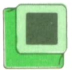
Otro comando que se puede utilizar con ficheros es

BOARCHIVO «nombrefich

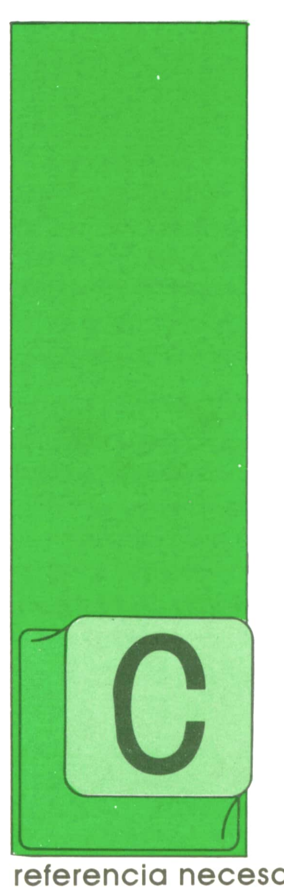
(abreviatura de BOrra ARCHIVO) que sirve para borrar el fichero que se llama *nombrefich* de la cinta o diskette.

Es bueno usar este comando para borrar ficheros que contengan definiciones de procedimientos que sabemos que no nos valen porque las hayamos modificado después o porque no tengamos que volver a utilizarlas. De esta forma, dejaremos espacio libre en la cinta o diskette para guardar otros ficheros que sí vayamos a necesitar más tarde.

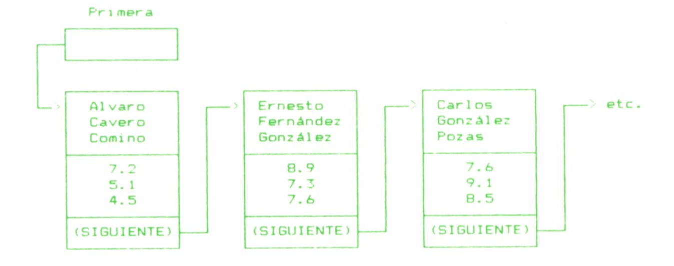
PASCAL



Listas encadenadas



COMO ya hemos visto, se podría definir una «lista encadenada» (también denominada «lista lineal») como una estructura de datos en la que cada elemento contiene la referencia necesaria para acceder al siguiente. Estas estructuras son de una enorme importancia en multitud de aplicaciones de proceso de datos; recordemos el ejemplo que vimos:



La definición de estas fichas podría ser:

```
type
   Nombre_t = array [1..12] of char;
   Punt_t   = ^Ficha_t;
   Ficha_t  = record
                Nombre,
                Apellido1,
                Apellido2 : Nombre_t;
                Nota_A,
                Nota_B,
                Nota_C    : real
                Siguiente : Punt_t
              = end;
var
   Primera: Punt_t;
```

La variable Primera sería necesaria para poder acceder a la primera ficha de la lista; por otra parte, el campo Siguiente de la última ficha debería tener el valor estándar NIL.

Al definir el tipo Punt_t se ha hecho referencia al tipo Ficha_t que todavía no estaba definido; éste es uno de los pocos casos en que eso está permitido (el compilador puede aceptarlo porque, independientemente de cuáles sean los tipos de variable a los que vayan a apuntar, todos los punteros tienen internamente la misma estructura y con la definición tal como está ya se puede saber que Punt_t es algún tipo de puntero). Si se hiciera primero la definición de Ficha_t, como en ésta se hace a su vez referencia al tipo Punt_t que todavía no estaría definido, se produciría un error.

Algunos compiladores no toleran esta situación, por lo que habría que definir Ficha_t de la siguiente manera:

```
type
   Nombre_t = array [1..12] of char;
   Ficha_t = record
               Nombre,
               Apellido1,
               Apellido2 : Nombre_t;
               Nota_A,
               Nota_B,
               Nota_C    : real
               Siguiente : ^Ficha_t;
             end;
 var
   Primera: ^ Ficha_t;
```

Con estos mismos compiladores también suele suceder que, como el campo Siguiente y otros posibles punteros (como

Primera) no se pueden declarar utilizando la misma definición previa, son conceptuados como de distinto tipo y no es posible asignarlos unos a otros; en el caso de Primera podríamos, por ejemplo, utilizar una variable AntesQuePrimera de tipo Ficha-t cuyo campo Siguiente se utilizaría para apuntar a la primera ficha, desperdiciándose todos los demás campos.

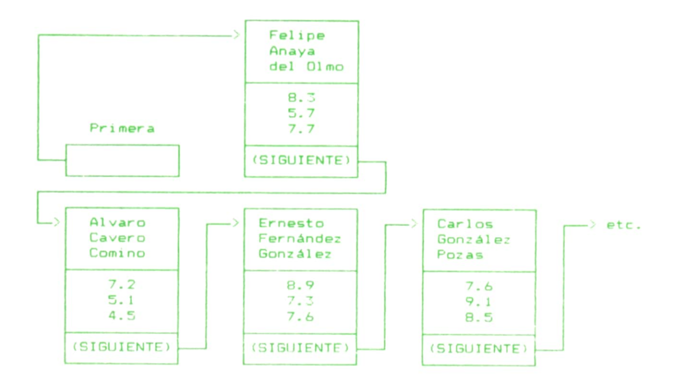
Vamos a ver algunas de las operaciones más elementales que se pueden realizar con listas encadenadas:

— Inserción de un nuevo elemento al comienzo de la lista

Esta es, probablemente, la operación más sencilla de todas. Lo primero que habría que hacer sería crear con NEW la nueva ficha y opcionalmente rellenarla de datos, utilizando para referirnos a ella un puntero auxiliar P de tipo Punt-t; tras ello, su campo Siguiente debería tomar el valor de Primera (para apuntar así a la hasta entonces primera ficha) y, por último, Primera habría de pasar a apuntar a la nueva ficha, es decir, debería tomar el valor de P:

```
new (P);
(* Rellenar P^... *)
P^.Siguiente := Primera;
Primera := P;
```

La lista del ejemplo una vez ampliada podría ser:



Para crear una lista partiendo de cero a base de insertar elementos por delante, bastaría con empezar creando una lista vacía y repetir la secuencia anterior para las diferentes fichas. Una lista vacía sería aquélla en que Primera no apuntase a ninguna ficha y, por tanto, se crearía así:

```
Primera := nil;
```

Al crear una lista de esta manera, cuanto más tarde se añadiese una ficha, más adelantada quedaría. Para conseguir que el orden de crecimiento de la lista fuese el contrario, habría que ir añadiendo los elementos justo por el final.

— Recorrido secuencial de una lista

Imaginemos que hubiese que recorrer una lista para procesar todos sus elementos, empezando por el primero; si «Actual» fuese una variable de tipo Punt-t que apuntase en cada momento a la ficha que se fuese a procesar, la secuencia adecuada sería:

```
(* Empezar por la primera: *)
Actual := Primera;
(* Si hay ficha, procesarla: *)
while Actual <> nil do
  begin
    Procesar (Actual^);
    (* Pasar a la siguiente: *)
    Actual := Actual^.Siguiente
  end;
```

Si hubiese que recorrer la lista en busca de un elemento dado, haríamos lo mismo pero poniendo como condiciones de salida del bucle la llegada al final o el hallazgo de la ficha:

```
Actual := Primera
while (Actual <> nil)    and
    not "es Actual^" do
        Actual := Actual^.Siguiente;
if Actual <> nil then (* es Actual^ *)
```

«es Actual^" sería la condición booleana que comprobaría si Actual^ es el elemento buscado. Si no existiese ese elemento, llegaría un momento en que Actual tendría el valor NIL, y tal como está programada la salida del bucle, se evaluaría la condición de identificación a pesar de ello, lo cual sería erróneo, pues, lógicamente, esa evaluación implica analizar el contenido de uno o más campos de Actual^; es exactamente el mismo problema que teníamos con las tablas de caracteres a la hora de determinar la longitud efectiva de un texto. Una forma correcta de programarlo sería, por tanto:

```
Actual := Primera;
EsEsta := false;
while (Actual <> nil) and not EsEsta do
  if "es Actual ^ " then EsEsta := true
   else Actual := Actual ^ .Siguiente;
if EsEsta then...
```

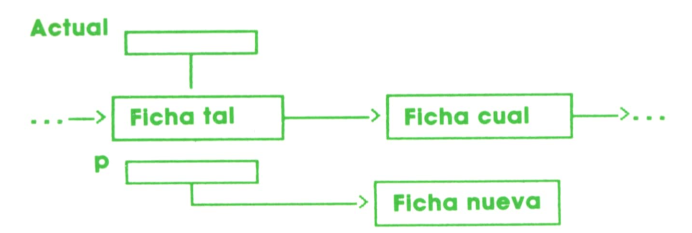
Avanzar sería una variable de tipo Boolean.

— Inserción de un nuevo elemento tras uno dado de la lista.

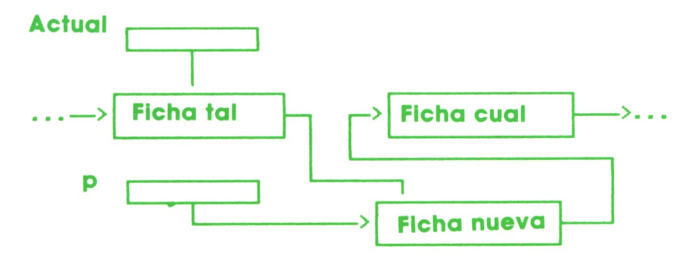
Esto es muy parecido a la inserción al comienzo de la lista: si Actual apunta al elemento de la lista tras el cual se desea insertar el nuevo, que está apuntado por P, haríamos:

```
new (P);
(* Rellenar P^... *)
P ^.Siguiente := Actual^.Siguiente;
Actual^.Siguiente := P;
```

o sea, hacemos que el campo Siguiente de P^ apunte a la ficha que va tras Actual^ y luego hacemos que tras Actual vaya P^ . Gráficamente, sería pasar de la situación:



a la situación:



— Inserción de un nuevo elemento delante de uno dado.

En principio, esto parece complicado pues habría que modificar el campo Siguiente de la ficha anterior a la actual para que apuntase a la nueva, y desde un ficha dada sólo podemos ir hacia delante, nunca hacia atrás. Sin embargo, se puede hacer de una manera muy sencilla, mediante un pequeño truco que consiste más o menos en insertar la nueva detrás de la actual —cosa que ya sabemos hacer— e intercambiar sus contenidos:

```
new (P);
P ^:= Actual^;
(* Rellenar Actual^ con *)
(* los nuevos datos     *)
Actual^ .Siguiente := P;
```

— Eliminación del sucesor de un elemento dado de la lista

Para eliminar el elemento siguiente a Actual^ de la estructura, basta con «saltar» por encima de él, o sea, hacer que el puntero Siguiente de Actual^ tome el valor del puntero Siguiente de la ficha posterior:

```
Proximo := Actual^ .Siguiente;
Actual^ .Siguiente := Proximo^ .Siguiente;
```

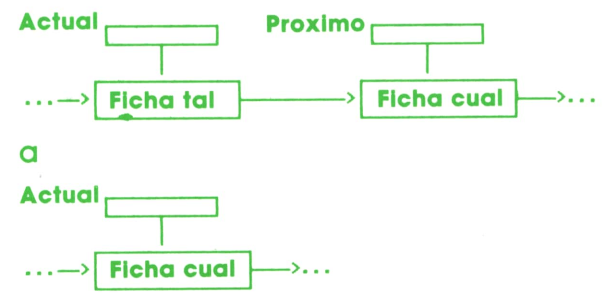
Próximo sería un puntero auxiliar. Por supuesto, la ficha Próximo^ seguiría en memoria tras su eliminación de la lista podríamos entonces incorporarla a otra lista o eliminarla de memoria, pero esto último veremos cómo hacerlo en otra ocasión.

— Eliminación de un elemento dado de la lista

Esto, al igual que la inserción por delante, parece difícil en principio, pues hay que retocar el puntero de la ficha anterior para que apunte a la siguiente, saltando por encima de la actual. De manera análoga a como se hizo entonces, podemos copiar el contenido de la siguiente en Actual^ y después borrar aquélla (aunque habría que comprobar antes que Actual^ no es la última de la lista):

```
Proximo := Actual^ .Siguiente;
Actual^ := Proximo^ ;
```

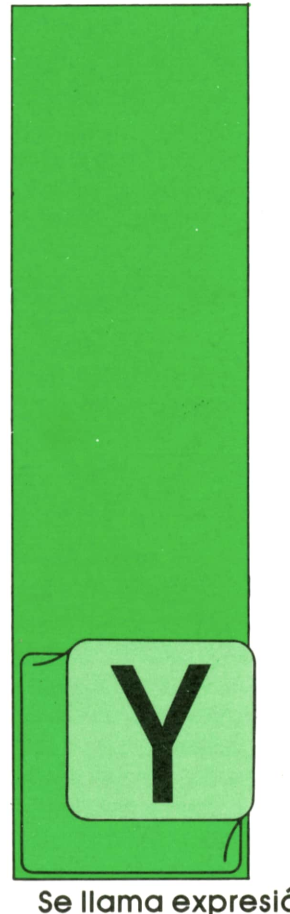
Es decir, pasaríamos de:



Y Próximo^ quedaría apuntado a una ficha que ya no pertenecería a la lista.

OTROS LENGUAJES

OPERACIONES ARITMETICAS FORTRAN (3)



A que el FORTRAN es el lenguaje orientado al cálculo científico por excelencia, es importante hacer hincapié en el tema de las expresiones aritméticas.

Se llama expresión aritmética a un conjunto de constantes, variables y referencias a funciones, relacionadas entre sí por paréntesis, símbolos de operación, etcétera.

Los operadores aritméticos que se pueden utilizar en FORTRAN son:

+: Suma
*: Multiplicación
–: Resta
/: División
**: Exponenciación

La evaluación de las expresiones aritméticas se hace de acuerdo con la siguiente jerarquía:

1. Se evalúan las funciones.
2. Se calculan las exponenciaciones (**).
3. Se evalúan los productos y divisiones (*, /).
4. Por último, se calculan las sumas y restas (+, -).

Si en una expresión aparecen varios operadores con la misma jerarquía, estas operaciones se efectuarán de izquierda a derecha.

El uso de paréntesis permite romper el orden de operación, ya que siempre se evalúan primeramente las expresiones que aparecen dentro de un paréntesis.

Algunos ordenadores imponen limitaciones a la hora de efectuar operaciones aritméticas entre valores de diferentes tipos. Resulta conveniente consultar el manual correspondiente.

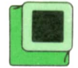
En los operadores explicados hasta ahora, el FORTRAN no se diferencia apenas de los restantes lenguajes. Lo que le proporciona su gran capacidad de cálculo son las funciones de librería.

El concepto de función en FORTRAN es similar al del álgebra y se pueden utilizar en expresiones aritméticas, sólo con poner el nombre de la función seguido de los argumentos entre paréntesis.

Alguna de las funciones de librería más conocidas son:

— Resto de la división: *MOD (I1, I2).*
— Raíz cuadrada: *SQRT (R).*

De todas maneras, es conveniente consultar el manual antes de emplear estas funciones, con el fin de evitar errores.



Expresiones lógicas

Se denomina expresión lógica a un conjunto de constantes, variables, funciones lógicas y expresiones de relación unidas entre sí mediante operadores lógicos.

En FORTRAN se entiende como expresión de relación al conjunto formado por dos expresiones aritméticas comparadas entre sí.

exp-1 OPERADOR exp-2

Los operadores de relación permitidos son:

.*EQ.* Igual. .*NE.* Distinto.
.*LT.* Menor. .*LE.* Menor o igual.
.*GT.* Mayor. .*GE.* Mayor o igual.

Si el resultado de la comparación es cierto la expresión de relación tendrá valor .*TRUE.*.

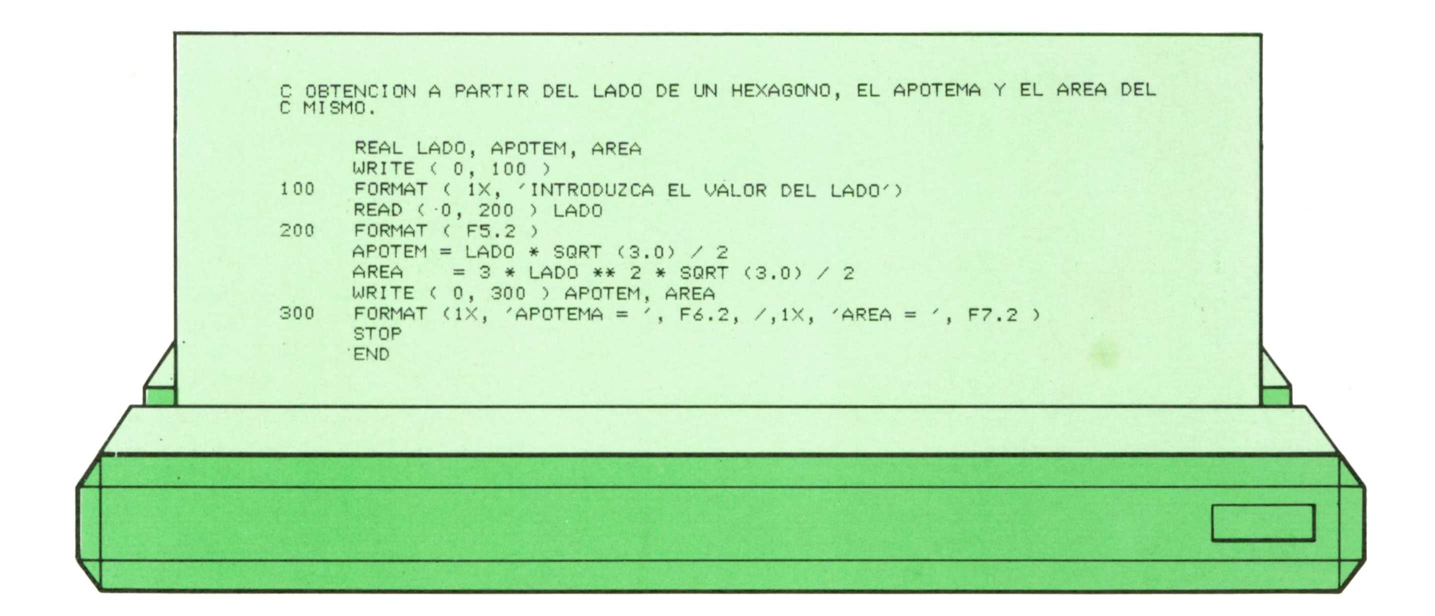
Las expresiones de relación pueden ser unidas mediante operadores lógicos.

— .*NOT.* Niega una expresión.

— .*AND.* Arrojará un resultado cierto (.*TRUE.*) sólo cuando las dos expresiones lógicas tienen valor .*TRUE.*.

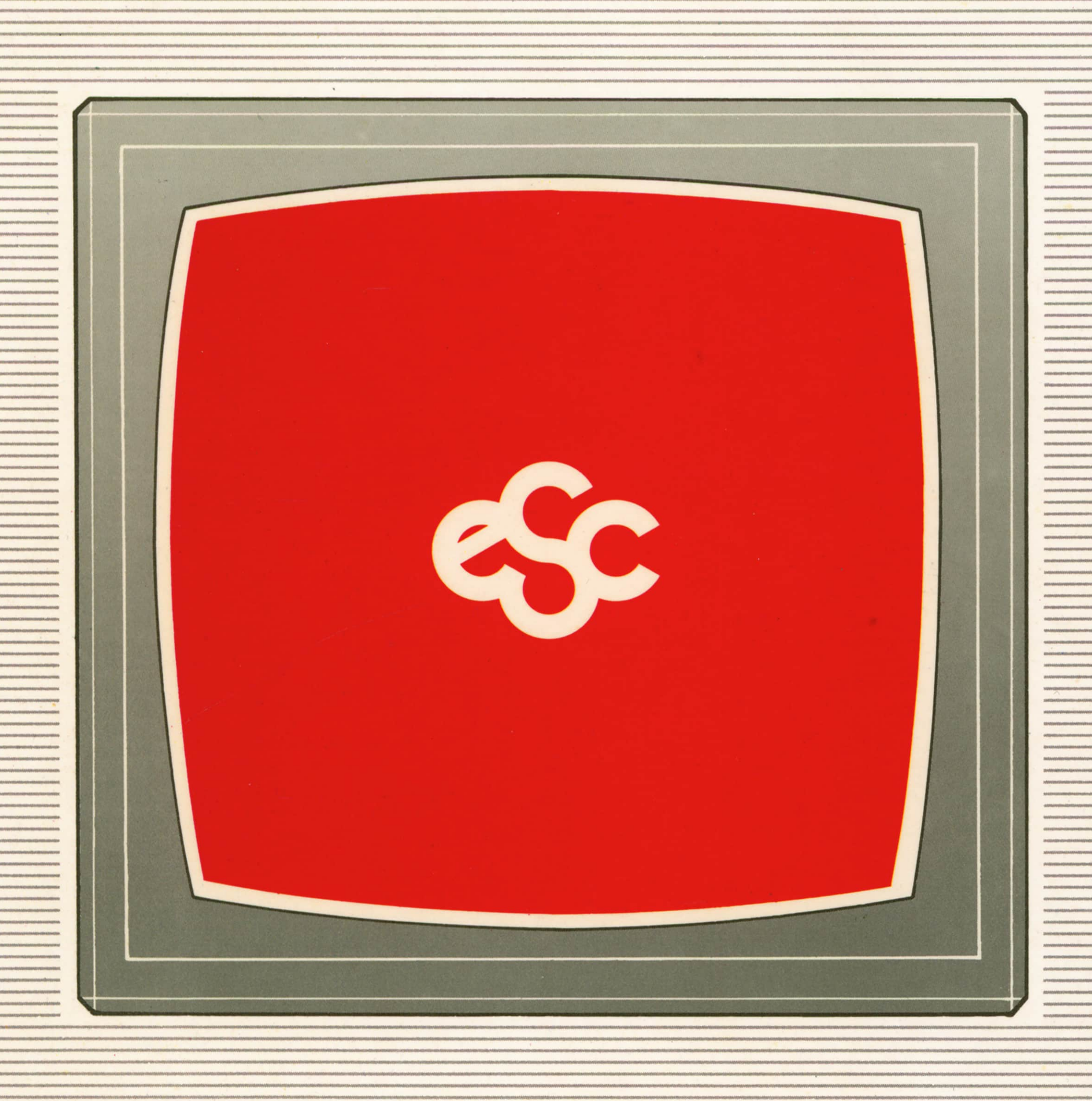
— .*OR.* Para que tenga valor .*TRUE.* basta con que sea cierta una de las dos expresiones que relaciona.

Las expresiones lógicas tienen una gran importancia en las sentencias condicionales que se verán más adelante.



```
C OBTENCION A PARTIR DEL LADO DE UN HEXAGONO, EL APOTEMA Y EL AREA DEL
C MISMO.

       REAL LADO, APOTEM, AREA
       WRITE ( 0, 100 )
100    FORMAT ( 1X, 'INTRODUZCA EL VALOR DEL LADO')
       READ ( 0, 200 ) LADO
200    FORMAT ( F5.2 )
       APOTEM = LADO * SQRT (3.0) / 2
       AREA   = 3 * LADO ** 2 * SQRT (3.0) / 2
       WRITE ( 0, 300 ) APOTEM, AREA
300    FORMAT (1X, 'APOTEMA = ', F6.2, /,1X, 'AREA = ', F7.2 )
       STOP
       END
```



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼